Departamento de Informática

INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

PROYECTO FIN DE CARRERA

APLICACIÓN DE TÉCNICAS DE APRENDIZAJE BASADO EN INSTANCIAS EN EL VIDEOJUEGO MARIO BROS.

Autor: Rafael Liaño Ruiz

Tutor: Raquel Fuentetaja Pizán
Director: Fernando Fernández Rebollo

Marzo 2014

Título: APLICACIÓN DE TÉCNICAS DE APRENDIZAJE BASADO EN INSTANCIAS EN EL VIDEOJUEGO MARIO BROS.
Autor: Rafael Liaño Ruiz
Tutor: Raquel Fuentetaja Pizán
Director: Fernando Fernández Rebollo

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día __ de _____ de 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

Agradezco a mi novia y familia todo el apoyo que me han dado para la realización de este proyecto.

A mis tutores por toda la ayuda prestada.

A todos los profesores con los que tanto he aprendido a lo largo de la carrera.

Resumen

Este proyecto se centra en analizar cómo se pueden aplicar los conocimientos de la Inteligencia Artificial (IA), en concreto el Aprendizaje Automático (AA) en el videojuego Mario Bros.

En los videojuegos, el jugador tiene el control de uno o varios personajes, equipo, automóvil, etc., que se manejan con una serie de acciones finitas. Mediante la combinación de estas, el jugador proporciona al sistema o personaje una inteligencia para que solucione los conflictos que se producen.

Con la industria de los videojuegos en pleno auge, el desarrollo de personajes inteligentes en entornos virtuales ha supuesto un cambio en la manera de desarrollar videojuegos, siendo éstos más complejos e introduciendo al usuario en mundos más cercanos a la realidad. Uno de los cometidos de la IA es humanizar los personajes no jugadores (PNJ) para crear la ilusión de ser controlados por humanos.

La competición de MarioAI ofrece una interfaz en la que se pueden aplicar diversos algoritmos para construir un agente automático que sea capaz de mover al personaje Mario a través de diferentes niveles. A lo largo del proyecto se analiza y se crea un agente automático mediante el uso de AA con técnicas de aprendizaje basado en instancias (IBL). Para ello, el jugador humano, mediante la repetición de varias partidas en el juego, genera automáticamente acciones a lo largo del nivel mientras lleva a Mario a la meta. Dichas acciones se almacenan junto con la información respectiva del nivel para su posterior uso en el agente automático.

En siguientes capítulos se introducen los conceptos principales de la IA y se detallan los algoritmos y técnicas que se han aplicado en la creación del agente para el videojuego MarioAI.

Palabras clave: Mario Bros, inteligencia artificial, aprendizaje automático, aprendizaje basado en instancias, clustering.

Abstract

This Project focuses on analysing how Artificial Intelligence (AI) knowledge, specifically Machine Learning (ML), can be used in the Mario Bros video game.

Usually, in video games, the player controls one to several characters, team, car, etc., which are managed by a sequence of finite actions. If we combine these actions, the player conveys intelligence to the character or the system with the aim of solving problems which can happen.

Due to the apogee in the video games industry, intelligent characters development in virtual environments has resulted in changing the way video games are developed, making them more complex and giving the user the chance being introduced in a world which seems real. One of the aims of AI is making the non player characters (NPC) appear as they were human, to create the illusion of being controled by humans.

MarioAI championship offers an interface in which we can apply various algorithms to build an automated agent that is able to move the character Mario through different levels. During the Project, an automated agent is analysed and created by using instance based learning (IBL). For this result, the human player through the repetition of various matches in the game, produces some actions automatically during the level, while Mario reaches the goal. These actions are stored with information related to the level for their subsequent use in the automated agent.

In the following chapter the main concepts of AI are introduced. Also, algorithms and technics applied in the agent for the MarioAI video game are detailed.

Keywords: Mario Bros, arificial intelligence, machine learning, instance based learning, clustering.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción y objetivos

Este primer capítulo introduce al lector en los temas principales del proyecto al igual que se definen los objetivos del mismo. Finalmente se presenta la estructura de esta memoria.

1.1 Introducción

Como seres inteligentes que somos, la evolución nos ha otorgado capacidades mentales superiores a otras especies. La palabra inteligencia proviene del latín *intellegere*, compuesto de *inter* "entre" y *legere* "leer, escoger", por lo que, etimológicamente, inteligente es quien sabe escoger. La inteligencia permite elegir las mejores opciones para resolver un problema. Con su uso se ha conseguido crear sistemas informáticos capaces de realizar cómputos a una velocidad que ningún humano podrá alcanzar, y gracias a estos sistemas, mediante programación, se pueden dotar de inteligencia artificial.

La informática es el medio que se utiliza para generar esa inteligencia, como lo hacen los robots, máquinas en sistemas de producción y programas informáticos. Basan su conocimiento en técnicas y algoritmos especialmente diseñados para éste propósito y realizan tareas que un humano no es capaz de hacer. De esta manera se crean, por ejemplo, brazos mecánicos que son capaces de soldar un chip o colocar componentes en un determinado lugar, tan pequeño que no se podrían obtener los mismos resultados con

nuestras propias manos. Existen robots capaces de moverse, hablar e interactuar con otra persona dando la sensación de ser un ser vivo e inteligente. Se puede determinar si una pieza es defectuosa con el uso de la visión artificial. En todas estas situaciones se habla de cierta inteligencia que el humano ha aplicado a la máquina.

En el mundo de los videojuegos sucede lo mismo, ya que se trata de un sistema en el que el usuario interactúa con un personaje en un entorno que contiene otros personajes o rivales que son controlados con inteligencia artificial. De este modo se crea una relación entre el usuario y la máquina que proporciona una idea de realismo, imaginando que se interactúa con otro usuario real.

En este proyecto, mediante el uso de esa inteligencia, se construye un agente automático para el videojuego MarioAI que pueda superar diferentes pantallas igual que podría hacer cualquier jugador. En el siguiente capítulo se detallan los objetivos para lograr su consecución.

1.2 Objetivos

Para generar el agente automático se utiliza la versión MarioAI del videojuego Mario Bros, siendo el mismo videojuego reescrito en Java y utilizado para desarrollar diferentes algoritmos con el fin de aprender diversas formas de superar los niveles del juego. Dado que existen numerosas soluciones para su consecución, en base a los múltiples algoritmos que se pueden utilizar como A*, aprendizaje por refuerzo, redes de neuronas, etc., se ha decidido utilizar el aprendizaje basado en instancias para su desarrollo ya que genera soluciones a través de la información suministrada por el usuario. Además no se ha encontrado información sobre su uso en la competición MarioAI y sirve como método de ampliación en el conocimiento de esta técnica.

El objetivo principal del proyecto es implementar un agente mediante técnicas de AA basado en IBL, que sea capaz de jugar automáticamente reproduciendo el comportamiento humano, cuyos datos de entrenamiento se han generado anteriormente. Para alcanzar este objetivo se plantean los siguientes objetivos secundarios:

- Estudiar el código proporcionado por la competición de MarioAI para comprender su funcionamiento e implementar el agente automático.

- Diseño de la solución al problema mediante el uso de IBL y su posterior implementación.

- Realizar experimentos para evaluar la aproximación y determinar qué combinación de parámetros es la más adecuada a la hora de ejecutar el algoritmo de aprendizaje automático.

- Documentar la aproximación con la presente memoria para una mejor comprensión del proyecto realizado.

1.3 Estructura de la memoria

A lo largo de este documento, se pueden encontrar las siguientes secciones:

- Capítulo 1 – Introducción y objetivos:

  Breve introducción del proyecto, objetivos que pretende alcanzar y la estructura del documento.

- Capítulo 2 – Estado de la cuestión:

  En este capítulo se expone una descripción de los aspectos que se van a relacionar y aplicar en el proyecto. Se definen diversos conceptos de inteligencia artificial y aprendizaje automático, se trata la historia del personaje Mario y se detallan los programas MarioAI y Weka que se utilizan para crear el agente automático.

- Capítulo 3 – Análisis y diseño del sistema propuesto:

  Se realiza un análisis de la aproximación y la implementación del agente automático, detallando el proceso para su funcionamiento y las entradas y salidas de las que se compone así como todo lo necesario para su ejecución.

- Capítulo 4 – Diseño de experimentos y resultados:

  En esta sección se incluyen todos los experimentos realizados para comprobar el funcionamiento del agente automático.

- Capítulo 5 – Conclusiones y líneas futuras:

  Se recogen las conclusiones que se obtienen durante el proyecto y se reflexiona acerca del cumplimiento de los objetivos. Además, se citan posibles líneas de investigación para ampliar el estudio realizado.

- Capítulo 6 – Gestión del proyecto:

  Se detallan las fases de desarrollo del proyecto, los medios empleados, la planificación y los costes asociados al mismo.

- Glosario:

  Resumen de los acrónimos utilizados en la memoria.

- Referencias:

  Bibliografía que se ha consultado y hecho referencia.

Capítulo 2

Estado de la cuestión

En el siguiente capítulo se explican los conceptos más importantes que se han tratado en el proyecto, se detallan los algoritmos y el videojuego utilizado, así como la historia de su personaje principal.

2.1 Inteligencia Artificial

El desarrollo de la inteligencia artificial supone una evolución en todos los aspectos, tanto emocionales como constructivos, creando así nuevas tecnologías y herramientas disponibles. Con el impulso de la tecnología y los ordenadores se han creado programas y máquinas capaces de realizar tareas mucho más rápidas, con elementos cada vez más pequeños y en tiempos más cortos. Todo ello sería imposible para el hombre ya que no se podrían manipular objetos tan pequeños en tan poco tiempo ni con tan pocos recursos.

Con las nuevas técnicas y herramientas se necesita controlar dichas máquinas para que hagan la tarea encomendada de la manera para la que han sido configuradas. Es necesario entonces dictar ciertas órdenes y cada vez de una manera más inteligente, de forma que pueda seguir funcionando y aprendiendo a medida que se avanza en el tiempo.

Se crean así programas inteligentes que son capaces de validar piezas en una fábrica con visión artificial, interactuar en un juego con el ordenador como si fuera un usuario más e incluso crear máquinas y tecnologías inteligentes en campos de la robótica o

medicina. Es por tanto una rama muy importante que se seguirá desarrollando en el futuro con grandes avances.

En el presente proyecto se utilizan técnicas de IA para crear un agente capaz de jugar al videojuego MarioAI mediante un agente automático. En los videojuegos se utiliza enormemente la IA teniendo principalmente un rival siempre disponible al que enfrentarse ya sea en batallas, carreras u otros tipos de juegos, con diferentes niveles de dificultad y con sistemas cada vez más potentes.

A continuación se describen las técnicas de AA aplicadas en este proyecto.

2.2 Técnicas de clustering o agrupamiento

Un clúster es una colección de objetos que son similares entre sí. Las técnicas de clustering se encargan de realizar una partición de un conjunto en varios grupos o categorías. La técnica de clustering en éste proyecto ha sido k-medias y se ha utilizado para ello el algoritmo SimpleKMeans de la herramienta Weka.

K-medias es una técnica que separa los ejemplos de entrenamiento en un conjunto de clases que agrupan a los elementos más parecidos. El número de clases a crear es proporcionado por el usuario. El objetivo de la agrupación es obtener una partición de un conjunto de ejemplos que maximice la similitud entre los elementos de cada clase y minimice la similitud entre diferentes clases.

Las entradas se componen de un conjunto de ejemplos de entrenamiento, el número de clases que se desean obtener y la medida de similitud (ver *más adelante*). La salida será un conjunto de *k* clases que agrupen los ejemplos de acuerdo a dicha medida.

El espacio de problemas puede describirse como:

- Conjunto de estados: cada estado será una agrupación de los ejemplos en *k* clases. El conjunto de estados será el conjunto de todas esas posibles agrupaciones.

- Conjunto de operadores: el único operador es asignar un ejemplo a una clase.

- Estado inicial: se escoge aleatoriamente un estado inicial, seleccionando *k* ejemplos que formarán las semillas de cada clase.

- Metas: aquella agrupación de ejemplos en clases que maximiza la medida de similitud entre los ejemplos de las clases.

- Heurística: introducir cada ejemplo en la clase en la que se encuentra la semilla más parecida al que se quiere clasificar.

Su funcionamiento es el siguiente: Inicialmente para cada una de las k clases selecciona aleatoriamente un ejemplo como semilla o prototipo inicial de la clase. A partir de ahí realiza una serie de iteraciones hasta que convergen las clases. En cada iteración, agrupa cada ejemplo en la clase que contiene la semilla que más se le parece. Al final de cada iteración, recalcula la semilla de cada clase, como el punto medio (centroide) de todos los ejemplos que están agrupados en la misma clase.

En el proyecto, realizar este proceso de agrupación en clases similares sirve para que la búsqueda posterior de las instancias sea más acertada, ya que acotamos dicha búsqueda a un grupo más reducido y el tiempo de este proceso disminuye. Para seleccionar el número de clases se realizan pruebas con diferentes valores, como puede verse en el capítulo 4 *Diseño de experimentos y resultados,* para tratar de determinar qué valor es el más adecuado.

2.3 Aprendizaje basado en instancias

Esta técnica de aprendizaje supervisado consiste en guardar las instancias de entrenamiento de manera que para clasificar una nueva instancia simplemente se elige la clase más parecida o las K más parecidas, que anteriormente se han agrupado en el proceso de clustering. Para ello se maneja la distancia entre ejemplos.

Su uso en el proyecto es fundamental. El usuario realiza una primera fase de generación de datos de entrenamiento en la que debe jugar varias partidas al mismo nivel. Las acciones que ejecuta son almacenadas junto con el estado del juego como nuevas instancias. Por tanto una instancia está compuesta del par estado-acción. Una vez se obtienen todas las instancias generadas se procede a realizar un agrupamiento de las mismas en conjuntos similares.

Tras haber realizado el tratamiento a los datos de entrada o instancias, aprendizaje, cuando se ejecuta el agente automático, éste necesita conocer qué acción debe ejecutar. Para ello se procede a buscar la clase a la que pertenece el estado actual del agente y obtener de ella las instancias más similares (vecinos más cercanos, KNN). Tras ello se obtiene la acción de la instancia y el agente automático realiza su movimiento.

El algoritmo KNN, del inglés *K Nearest Neighbours,* se basa en que dado un conjunto de entrenamiento y una instancia a clasificar, los vecinos más cercanos serán obtenidos utilizando la distancia euclídea sobre la totalidad del conjunto.

La distancia euclídea se calcula con la siguiente ecuación:

$$d_e(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

También se podrían utilizar otras medidas de distancia, entre ellas la Manhattan, la cual en este caso no se ha utilizado debido a que en este proyecto se miden propiedades similares.

Uno de los principales valores es la elección del número de vecinos más cercanos que se desean obtener (K), ya que a través de ellos se consigue la acción que ejecutará el agente automático. Se utiliza un valor de K que varía entre 1 y 3, como puede verse en el capítulo 4 *Diseño de experimentos y resultados*. Uno de los problemas que puede surgir al usar la distancia euclídea para calcular los vecinos más cercanos, es la irrelevancia de algunos valores de sus atributos. Para subsanar este problema se le ha dado más importancia a los atributos más relevantes del proyecto, que son el estado y la acción de *Las instancias*.

2.4 Weka

La herramienta Weka, del inglés *Waikato Environment for Knowledge Analysis*, es una plataforma software para aprendizaje automático y minería de datos, distribuido bajo licencia pública general y se usa en investigación, educación y aplicaciones.

Sus funciones se basan en preprocesamiento de datos, agrupamiento o clustering, clasificación, regresión, visualización y selección. Está programado en Java y orientado a la extracción y análisis de resultados de datos con grandes cantidades de información. Incluye una interfaz gráfica de usuario para acceder y configurar las diferentes herramientas integradas. Las librerías de las que se compone el programa pueden ser llamadas desde la misma interfaz o desde clases propias en Java. El único requisito del programa es que los datos a analizar se almacenen con un cierto formato, pudiendo ser el propio del programa, conocido como ARFF (Attribute-Relation File Format), o bien otro formato importable como CSV (Comma-Separated Values).

En el presente proyecto se ha utilizado este programa para realizar el agrupamiento de las instancias en clases o grupos mediante diferentes números de clúster. Los algoritmos de clustering buscan grupos de instancias con características similares según un criterio de comparación entre valores de atributos de las instancias definidos en los algoritmos. Permite ignorar atributos que no interese considerar para el agrupamiento, de manera que el análisis de parecido entre instancias no considera los atributos seleccionados.

Una vez se realiza el agrupamiento, a las nuevas instancias que se generan a la hora de ejecutar el agente automático se les asigna uno de los grupos con el fin de obtener los k vecinos más cercanos. La finalidad será obtener la acción de estos. Para ello se utiliza el algoritmo KNN de Weka.

2.5 Mario y Nintendo

Mario es un fontanero que nació en 1985. Fue creado por Shigeru Miyamoto para la compañía Nintendo y ha sido un rotundo éxito durante todos estos años y continúa siéndolo en las últimas generaciones de consolas de la marca. El propósito de Mario es salvar a la Princesa Peach superando diferentes niveles de pantalla y enemigos. Se trata

de uno de los primeros videojuegos de plataformas de desplazamiento lateral que se convirtió en un rotundo éxito.

Figura 1. Super Mario Bros.

La empresa japonesa Nintendo se fundó hace más de cien años por Fusajiro Yamauchi y se dedica al sector de los videojuegos y la electrónica de consumo para el entretenimiento.

Su primer acercamiento al mundo lúdico de las recreativas se realizó con el título "Mario Bros." Los hermanos Mario y Luigi irán avanzado niveles eliminando los enemigos que salen por tuberías. Después vendrá "Donkey Kong", *Figura 2*, en 1981, un nombre más que parecido al del gran gorila ficticio *King Kong*, que mezclado con el marinero *Popeye* dio como resultado un juego en el que se trata de rescatar a la dama capturada. El resultado fue un éxito en Norteamérica y Japón, situando a Nintendo en una posición destacada en el mercado con respecto a sus competidores.

Cuando la compañía lanzó la primera consola de videojuegos NES (Nintendo Entertainment System) terminó de popularizar los personajes antes mencionados. El título "Super Mario Bros", *Figura 1*, el mayor éxito, junta a los hermanos otra vez con la finalidad de rescatar a la princesa del Reino Champiñón que ha sido secuestrada por Bowser, rey de los Koopas. Ha sido reconocido como uno de los videojuegos más vendidos de todos los tiempos, siendo incluso el primer gran récord de ventas tras la crisis de la industria de los videojuegos en 1983.

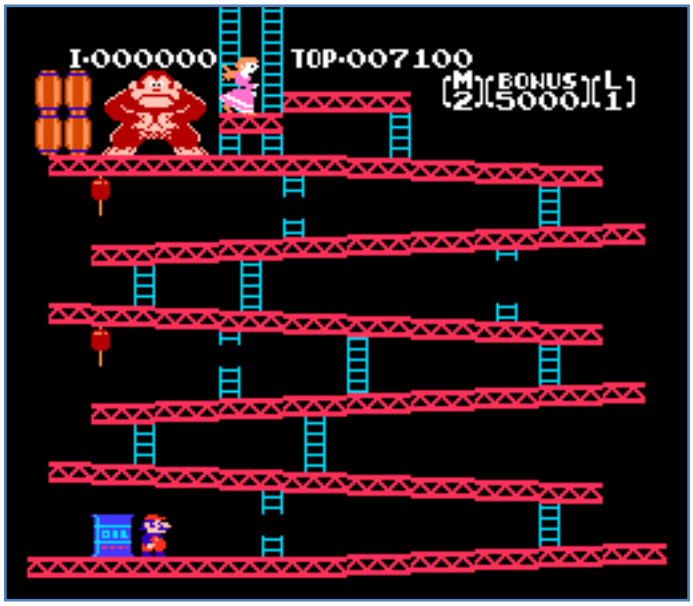

Figura 2. Donkey Kong.

2.6 MarioAI

La competición MarioAI, del inglés *Mario Artificial Intelligence,* consiste en realizar un agente que sea capaz de superar el mayor número de niveles posibles. En este proyecto sólo se utilizará la herramienta para realizar el agente automático, obviando la competición, la cual proporciona principalmente dos funciones con las que trabajar:

- getAction(), devuelve la acción a ejecutar por el agente. Se trata de 6 valores correspondientes a cada posible tecla o movimiento (izquierda, derecha, arriba, abajo, saltar y correr/disparar). Cada uno puede tomar dos valores: verdadero o falso.

- integrateObservation(), proporciona la información del estado. Se trata de una matriz con toda la información de la pantalla que se tiene en el estado actual. Con ella se puede guardar dicha información para tratarla posteriormente y devolver así una acción.

El juego, *Figura 3*, consiste en mover al personaje, mediante los diferentes movimientos posibles, al final de cada nivel atravesándolo de izquierda a derecha. La presencia de distintos enemigos así como fosos o agujeros complica la partida. Mario puede estar en uno de los siguientes modos:

- Fuego (Fire): Puede lanzar bolas de fuego para eliminar a los enemigos y saltar sobre ellos para eliminarlos.

- Grande (Big): Puede saltar encima de los enemigos para eliminarlos.

- Pequeño (Small): Igual al anterior.

Figura 3. MarioAI.

En cuanto nuestro personaje es alcanzado por algún enemigo pasará a un estado menor siendo el orden Fuego > Grande > Pequeño > Eliminado. Las opciones en las que Mario será eliminado de la partida son las siguientes:

- Estando en modo Pequeño y alcanzado por un enemigo.
- Mario cae en un foso o agujero al vacío.
- El tiempo de la partida se termina.

Existen ciertos elementos con los que Mario puede interactuar en el escenario, tenemos así:

- Monedas: Mario irá coleccionando monedas a lo largo de cada escenario.
- Bloques e ítems: Mario puede destruir bloques saltando desde abajo hacia arriba y golpeándolos con el puño. Existen bloques que contienen monedas o diferentes ítems como setas (Mario pasa de Pequeño a Grande) y flores (de Grande a Fuego).
- Enemigos: Puede eliminarlos saltando encima o disparando bolas de fuego. Tenemos diferentes enemigos como Goombas (seta marrón), tortugas (al saltar encima, Mario puede utilizar su concha para eliminar a otros enemigos), plantas carnívoras (solo se pueden destruir disparando), balas, enemigos que pueden volar, otros con pinchos, etc.

En la *Figura 4* se representan los enemigos, bloques, monedas e ítems presentes en el juego.

Figura 4. Enemigos, bloques, monedas e ítems.

Capítulo 3

Análisis y diseño del sistema propuesto

En el presente capítulo se detallan los programas que se han utilizado para la realización del proyecto así como la manera de instalarlos y ejecutarlos. Se explica cómo funciona el programa MarioAI y las técnicas que se aplican.

El siguiente paso es por tanto detallar cómo realizar su construcción y en qué consiste cada fase del proceso. Los apartados sucesivos dotarán al lector de una visión global de su funcionamiento.

3.1 Desarrollo del proyecto

La competición en la que se basa este videojuego, consiste en realizar un agente que sea capaz de superar un nivel mediante el uso de las funciones que nos proporciona el videojuego MarioAI, siendo las más importantes "getAction()" para la acción que supone el movimiento de Mario, e "integrateObservation()" donde se obtiene toda la información del estado en el que se encuentra el personaje.

Se plantea utilizar técnicas de aprendizaje basado en instancias para construir un agente automático que reproduzca el comportamiento humano en el juego.

El desarrollo del proyecto se basa en tres fases:

- Generación de datos de entrenamiento y creación del fichero con los datos de las partidas que juegue el usuario.
- Aprendizaje: creación del clúster y sus ficheros correspondientes.
- Utilización de los resultados anteriores en la ejecución del agente.

Se comenta en los siguientes apartados la descripción de la aproximación y la de implementación y diseño de software.

3.1.1 Descripción de la aproximación

En este apartado se procede a describir a alto nivel el desarrollo del proyecto cuyo funcionamiento general, *Figura 5,* es el siguiente:

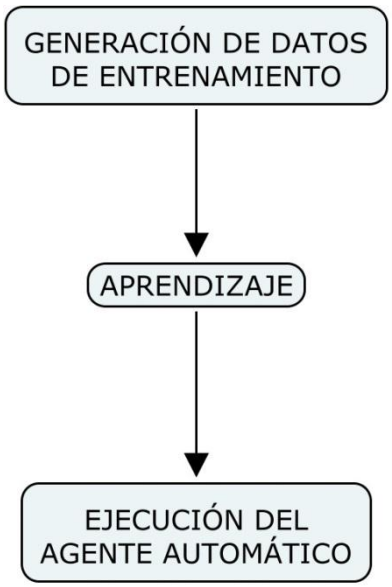

Figura 5. Descripción general.

Primeramente se generan los datos de entrenamiento, tarea del usuario o jugador, para crear las instancias. Para ello se debe jugar una serie de partidas en las que se guarda el par estado-acción. El estado corresponde a la situación en la que Mario se encuentra en el juego. La acción consiste en el movimiento que el usuario realiza con el personaje mientras juega. Cuantas más partidas se jueguen mayor será la cantidad de pares estado-acción que se almacenen.

También se guarda información adicional referente al estado que no ha sido utilizada en el proyecto. Esto es debido a que al realizar comparaciones y búsquedas posteriores sobre instancias no era necesaria tanta información y se centró el proyecto en simplemente utilizar el par estado-acción. Tampoco se ha eliminado ya que puede resultar

de utilidad para una posible ampliación del mismo. Los atributos no utilizados son los siguientes:

- Modo Mario: si Mario esta en modo Fuego (2), Grande (1) o Pequeño (0).

- Monedas que hay en la pantalla en ese instante: número de monedas que hay en el estado actual.

- Monedas recogidas por Mario hasta ese instante: recuento de las monedas que Mario ha recogido hasta el momento en el que se guarde la información.

- Detección de foso: indica si existe o no (1 o 0) un foso u hoyo por delante del personaje Mario por el cual podría caerse. Al estar Mario siempre en la celda [9,9], se comprueba que no exista nada desde la fila 7 a la 18 y desde la columna 10 a la 12, es decir, una matriz de (12x3). (ver *Información del estado (ventana)*)

- Detección de obstáculo: indica si existe o no (1 o 0) cualquier obstáculo por delante de Mario, desde la fila 7 a la 9 y desde la columna 10 a la 12, es decir una matriz de (3x3). (ver *Información del estado (ventana)*)

- Tiempo restante: tiempo que falta hasta llegar el contador a 0.

Las instancias, por tanto, se componen de la siguiente información:

- Información del estado (ventana): el juego proporciona la información de todo el estado actual. Se trata de una matriz de tamaño (19x19) en la que Mario siempre está centrado en la posición o celda [9,9]. Al tener tanta información para almacenar, se crea una ventana para obtener únicamente una pequeña parte de ella. Esta parte se ha denominado ventana y se trata de una matriz más pequeña que la original la cual el usuario puede definir su tamaño mediante las filas (inicial y final) y las columnas (inicial y final).

- Acción que se ejecuta en ese estado: la acción que el usuario ha realizado con el teclado a la hora de mover al personaje Mario en el juego. Una acción consta de seis posibles valores, uno correspondiente a cada tecla del juego. Se obtienen así las siguientes opciones:

    - Izquierda.

    - Derecha.

    - Abajo.

    - Arriba.

    - Saltar.

    - Correr/Disparar.

Como cada una de estas acciones puede tener dos valores posibles (verdadero o falso) y 6 acciones en total, dan como resultado un conjunto de 64 posibles acciones diferentes (2^6). Muchas acciones no tienen un significado válido ya que, por ejemplo, no tiene sentido el ir a la Derecha e Izquierda a la vez.

Para acotar el número de acciones válidas en este proyecto se ha realizado una definición manual de las combinaciones de teclas que generan acciones válidas, *Tabla 1*, que son las siguientes:

Número	Acción	Representación: (Izquierda, Derecha, Abajo, Saltar, Correr/Disparar, Arriba)
0	Izquierda	(1, 0, 0, 0, 0, 0)
1	Izquierda+Saltar	(1, 0, 0, 1, 0, 0)
2	Izquierda+Correr	(1, 0, 0, 0, 1, 0)
3	Izquierda+Correr+Saltar	(1, 0, 0, 1, 1, 0)
4	Derecha	(0, 1, 0, 0, 0, 0)
5	Derecha+Saltar	(0, 1, 0, 1, 0, 0)
6	Derecha+Correr	(0, 1, 0, 0, 1, 0)
7	Derecha+Correr+Saltar	(0, 1, 0, 1, 1, 0)

Tabla 1. Acciones válidas.

Por lo tanto, para que una instancia esté almacenada en el fichero de datos, tiene que haberse ejecutado una de las acciones válidas. Se descartan el resto de pares estado-acción cuya acción no está representada (no es válida).

Toda esta información representa una línea o instancia del fichero de datos. Esta primera parte representa la generación de datos de entrenamiento. Una vez completada, se obtiene un fichero de datos con todas las instancias guardadas y se procede a realizar agrupamiento o clustering sobre las instancias, fase de aprendizaje.

Inicialmente se prepara el fichero de datos en un formato compatible con la herramienta Weka. Para ello basta con añadir parámetros al inicio del fichero con el tipo de información de cada atributo y cambiar la extensión al fichero a ARFF para que éste sea compatible. Tras esto, se utilizan los algoritmos de Weka, en este caso el algoritmo k-medias (SimpleKMeans de la herramienta) para agrupar las instancias más parecidas en diferentes grupos en relación a un número de clúster que el usuario proporciona, generando así tantos ficheros como número de clústeres se hayan elegido. Todas las instancias se agrupan en estos ficheros y se guardan junto con el fichero modelo del agrupamiento, que almacena información sobre los centroides.

Por último la fase de ejecución del agente automático. Una vez todas las instancias estén agrupadas mediante clustering en diferentes ficheros, es el momento de ejecutar el agente automático. Para ello el funcionamiento del agente es el siguiente:

1. Obtener el estado actual del agente automático.

2. Calcular el clúster al que pertenece el estado. Para ello se utiliza el fichero modelo del agrupamiento.

3. Con el número del clúster, seleccionar el fichero de clúster que corresponda a dicho número y buscar los estados más parecidos al actual.

4. Se obtienen los K vecinos más cercanos y se extraen las acciones. Estas acciones se almacenan en una lista siendo todas ellas diferentes, es decir, sin repetición de acción. Esto se realiza ya que puede suceder que en el momento en el que el agente automático realice la acción no se produzca movimiento alguno y se pueda así ejecutar la siguiente acción de la lista. Se trata de evitar el bloqueo del agente automático.

5. Pasar la acción al programa MarioAI para que el agente realice la acción que se le proporcione.

3.1.2 Diseño software e implementación

Se procede a explicar más detalladamente la implementación, analizando cada fase del proceso, entradas/salidas y resto de información. A través de este apartado se describe la información necesaria para poder modificar todos los datos de entrada de cada fase mediante un ejemplo de ejecución del agente automático.

En la *Figura 6* se representa el funcionamiento y la estructura del proceso de generación de datos de entrenamiento. Mediante una serie de datos de entrada se genera un fichero de datos que se compone de todas las instancias, pares estado-acción. A medida que el usuario juega partidas se almacenan las instancias que sean válidas, es decir, que su acción esté en la lista de acciones válidas, ver *Tabla 1*.

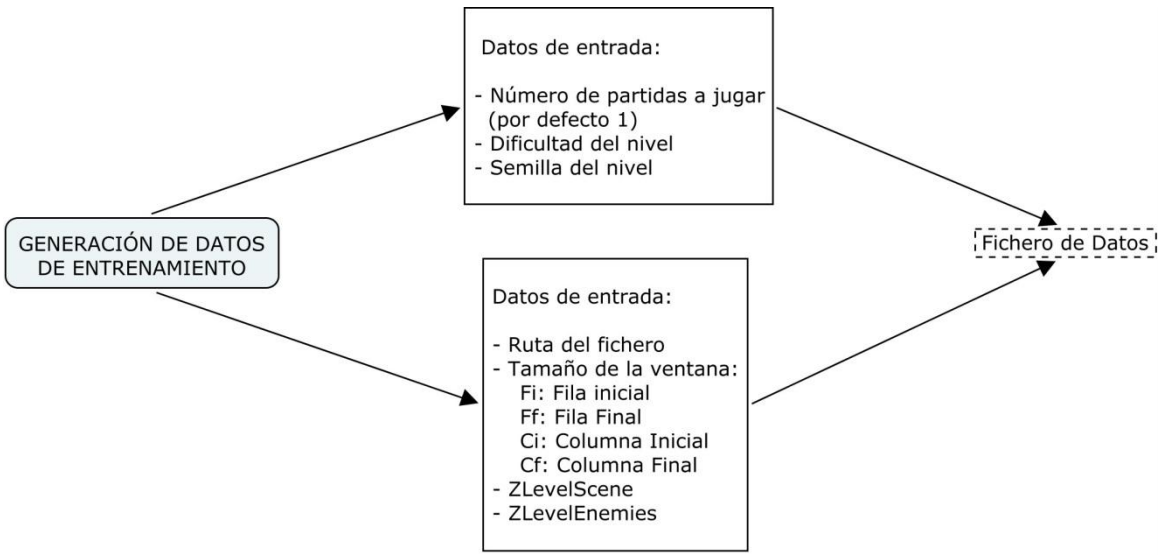

Figura 6. Entrenamiento

Los datos de entrada (Número de partidas a jugar, Dificultad nivel, Semilla del nivel Ruta del fichero, Tamaño de la ventana, ZLevelScene y ZLevelEnemies) para realizar el entrenamiento se modifican mediante los módulos "Main.java" y "HA.java".

Como se ha comentado en el apartado anterior (ver *Información del estado (ventana)*) y debido a que el espacio de estados en Mario es grande, se crea una ventana del tamaño que el usuario elija. Para ello basta con definir las casillas inicial y final tanto de las filas como de las columnas, teniendo en cuenta que la matriz principal que nos proporciona el programa es de tamaño 19x19 centrada siempre en Mario en la posición [9,9] de dicha matriz.

Una vez definidos estos parámetros, se puede modificar la ruta del fichero de datos y la profundidad con la que se quiera la información del estado de Mario, ZlevelScene y ZlevelEnemies, cuyos modos se pueden consultar en la página web *[14]* Cada uno de estos atributos del nivel puede ofrecer información más o menos detallada del estado actual dependiendo del valor que el usuario decida, pudiendo ser 0, 1 o 2. Arrojarán información más específica en los valores de estas variables cuanto mayor sea su número.

En el ejemplo de ejecución se procede a crear un fichero de datos, en el escritorio, cuyo nombre es "ejemplo.txt". La ventana de estados de Mario será de tamaño 3x3. En la *Figura 7* se muestran los parámetros a modificar para la generación de datos de entrenamiento.

Si se quiere modificar el número de partidas seguidas a jugar habrá que descomentar las líneas del módulo "Main.java" de MarioAI, y modificar el valor en rojo, ver *Figura 8.* Si se desea seleccionar diferente dificultad y semilla del escenario, modificar los valores en verde. Como se podrá comprobar en el capítulo 4, se pueden modificar las diferentes opciones de MarioAI mediante *argumentos*.

```
int zLevelScene = 1;
int zLevelEnemies = 0;
public FicheroEscritura escritura = new
FicheroEscritura("C:\\Desktop\\ejemplo.txt");
// Filas y Columnas (inicial y final) para la ventana de
observación.
// Mario siempre centrado en (9,9).
int Fi = 7;
int Ff = 9;
int Ci = 10;
int Cf = 12;
```

Figura 7. Código a modificar en HA.java.

```
   final BasicTask basicTask = new BasicTask(marioAIOptions);
//        for (int i = 0; i < 10; ++i)
//        {
//            int seed = 0;
//            do
//            {
//                marioAIOptions.setLevelDifficulty(i);
//                marioAIOptions.setLevelRandSeed(seed++);
//          }
    basicTask.setOptionsAndReset(marioAIOptions);
    basicTask.doEpisodes(1,true,1);
//        }
   System.exit(0);
```

Figura 8. Código a modificar en Main.java.

Por último, modificar la configuración y los argumentos para su ejecución en el programa principal (Eclipse). Para su configuración, ver

Figura 9, se selecciona el proyecto con el que trabajar, que en este caso se ha nombrado *MarioAI* y la clase principal *ch.idsia.scenarios.Main.*

El único argumento que se necesita es el del módulo de entrenamiento llamado "HA.java". Basta con escribir *-ag marioBros.agentes.HA,* ver *Figura 10*.

Una vez se ejecuta el código jugando una o varias partidas al mismo nivel se crea el fichero de datos con las instancias ("ejemplo.txt"), ver *Figura 11*.

Para comprender mejor que significa cada una de las instancias, en la *Tabla 2* se puede observar la información de la que se compone. Para ello se desglosa la primera línea de la *Figura 11*.

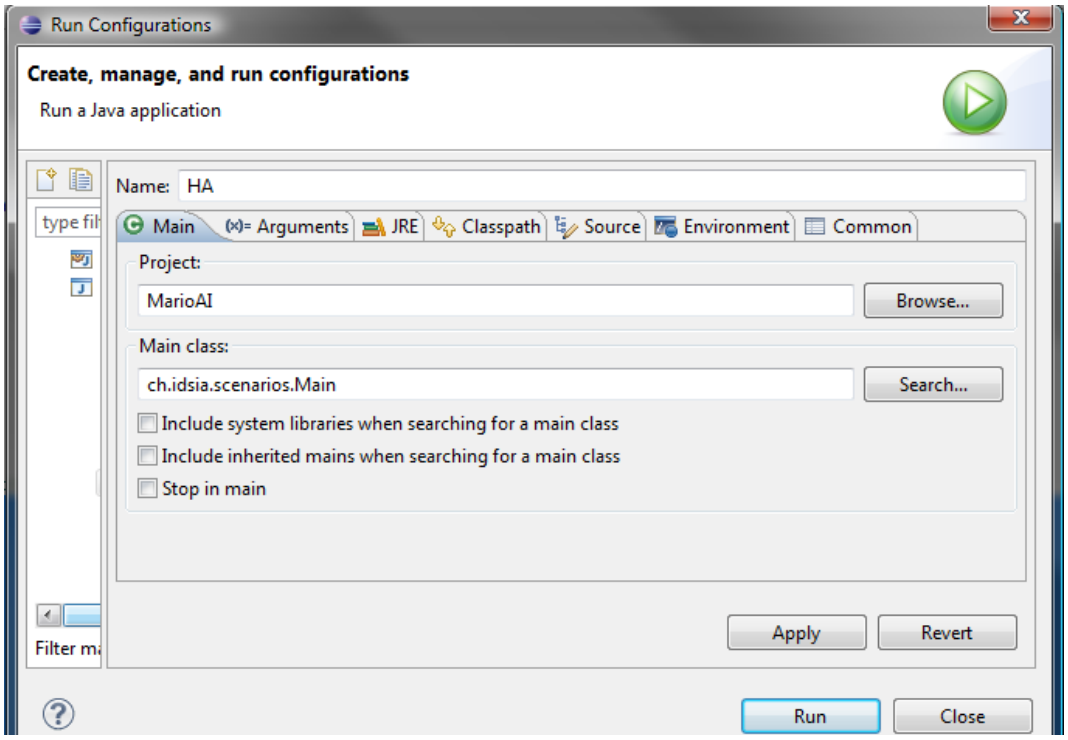

Figura 9. Cambio de configuraciones en Eclipse para HA.java.

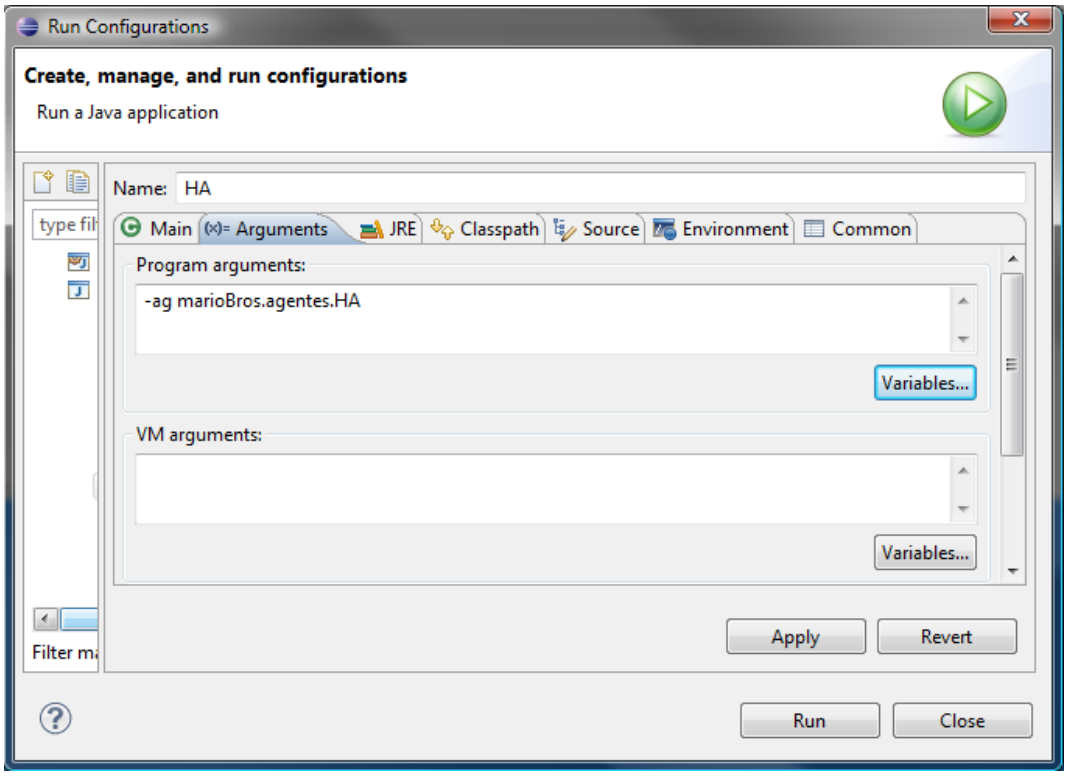

Figura 10. Cambio de argumentos en Eclipse para HA.java.

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 199
0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 198
0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 198
0, 0, 0, 0, 0, 0, 25, 0, 0, 2, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 198
0, 0, 0, 0, 0, 0, 25, 0, 0, 2, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 198
0, 0, 0, 0, 0, 0, 0, 25, 0, 2, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 198
0, 0, 0, 0, 0, 0, 25, 0, 0, 2, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 198
0, 0, 0, 0, 25, 0, 0, 0, 0, 2, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 198
0, 0, 0, 0, 25, 0, 0, 0, 0, 2, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 198
0, 0, 0, 0, 25, 0, 0, 0, 0, 2, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 198
0, 0, 0, 0, 25, 0, 0, 0, 0, 2, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 198
0, 0, 0, 0, 25, 0, 0, 0, 0, 2, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 198
0, 0, 0, 0, 25, 0, 0, 0, 0, 2, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 198
0, 0, 0, 0, 25, 0, 0, 0, 80, 2, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 198
0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 197
0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 197
0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 197
0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 197
0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 196
0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 196
0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 196
0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 196
0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 196
0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 196
0, 0, 0, 0, 0, 0, 0, 0, -85, 2, 0, 1, 0, 0, 0, 0, 0, 0, 1, 196
0, 0, 0, 0, 0, 0, 0, 0, -85, 2, 0, 1, 0, 0, 0, 0, 0, 0, 1, 196
0, 0, 0, 0, 0, 0, 0, 0, -85, 2, 0, 1, 0, 0, 0, 0, 0, 0, 1, 196
0, 0, 0, 0, 0, 0, 0, 0, -85, 2, 0, 1, 0, 0, 0, 0, 0, 0, 1, 196
…
```

Figura 11. Salida de HA.java, fichero ejemplo.txt.

Estado (FxC)	Modo Mario	Acción	Monedas Pantalla	Monedas Recogidas	Foso	Obstáculo	Tiempo Restante
0, 0, 0, 0, 0, 0, 0, 0, 0	2	0, 1, 0, 0, 0, 0	0	0	0	0	199

Tabla 2. Ejemplo de una instancia.

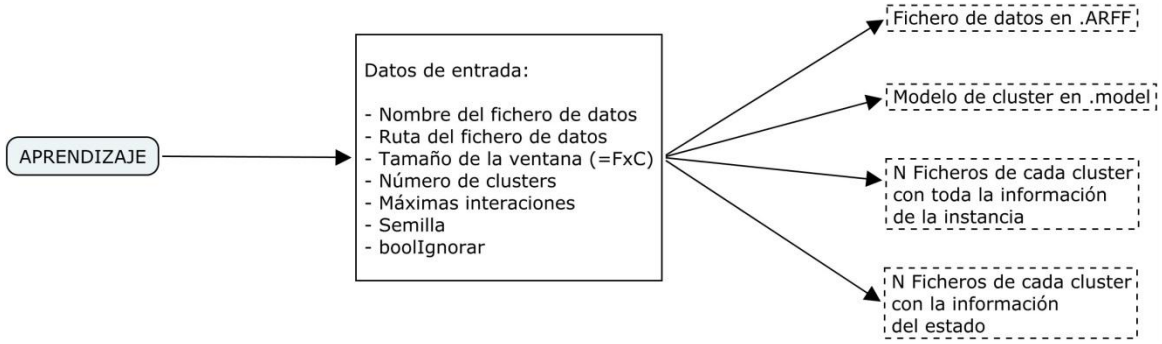

Figura 12. Aprendizaje.

Con el fichero de datos con las instancias creado en la fase de entrenamiento como entrada, el resto de datos de entrada a modificar en la fase de aprendizaje, ver *Figura 12,* son los siguientes:

- La ruta y nombre del fichero de datos creado en la generación de datos de entrenamiento.

- El tamaño de la ventana (un entero equivalente al tamaño de ésta = Filas x Columnas).

- Dependiendo del número de clústeres que se quieran utilizar y los diferentes valores de éste, modificar las variables "numClusters", "maxIteraciones", "semilla".

- Para ignorar los atributos que no se quieran usar en el proceso de clustering, activar el atributo "boolIgnorar". Éste booleano ignora los atributos que sean pasados por el String "strIgnorar" si está en valor Verdadero. Se utiliza esta acotación para dejar solo los estados y poder realizar consultas en la siguiente fase.

Estos parámetros se modifican en el módulo del fichero "Clustering.java", ver *Figura 13.*

```
//Nombre Fichero
public static String datos = "ejemplo";
//Ruta del fichero
public static String path = "C:\\Desktop\\";
//Tamaño Ventana = (M x N) de la ventana de Mario
public static int tamanoVentana = 9;
//Numero de Clústeres para SimpleKMeans
public static int numClusters = 3;
//Máximas iteraciones en SimpleKMeans
public static int maxIteraciones = 500;
//Semilla para SimpleKMeans
public static int semilla = 10;
//Si queremos ignorar o no los atributos
public static boolean boolIgnorar = true;
//String para seleccionar los atributos a ignorar
public static String strIgnorar = (tamanoVentana+1)+"-last";
                                  //deja solo los ESTADOS
```

Figura 13. Código a modificar en Clustering.java.

Para modificar la configuración para su ejecución en el programa principal, se selecciona el proyecto *MarioAI* y la clase principal *marioBros.parser.Clustering*, ver *Figura 14*. En este caso no es necesario ningún argumento.

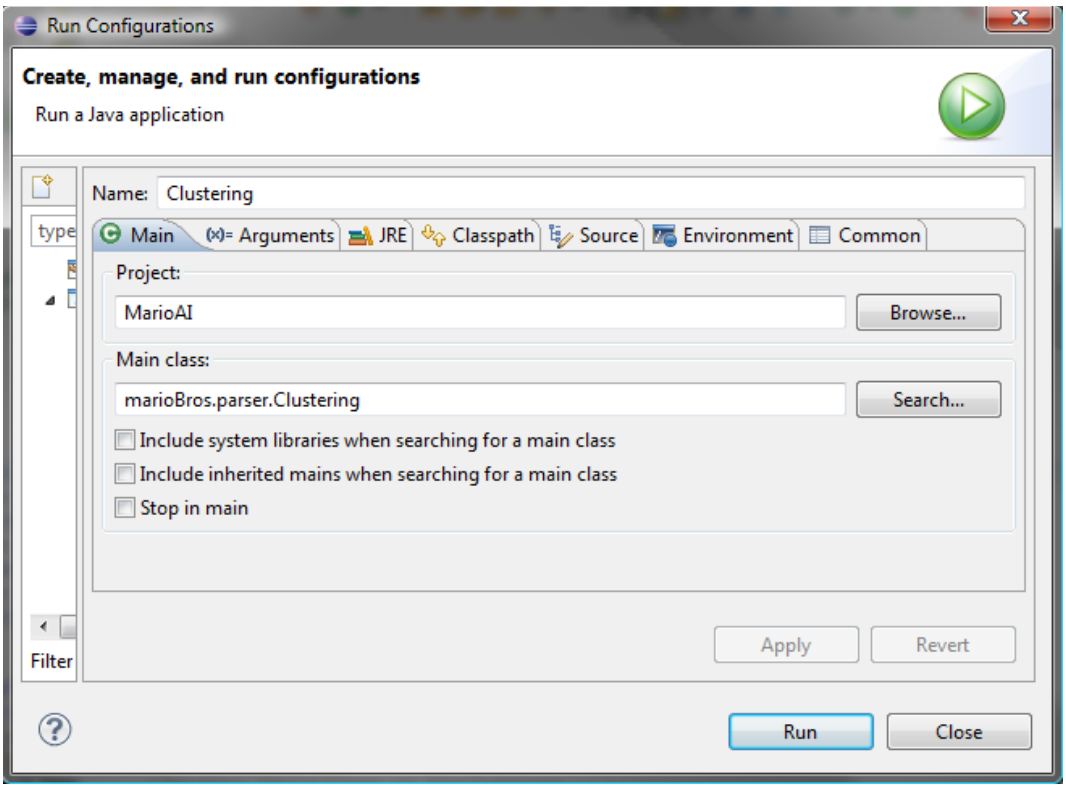

Figura 14. Cambio de configuraciones en Eclipse para Clustering.java.

Una vez decididos los parámetros se ejecuta el módulo, cuyo proceso es el siguiente:

- CrearFicheroArff(), para preparar el fichero de datos.

- clusteringSimpleKMeans(), para realizar el proceso de clustering mediante los algoritmos de Weka.

A continuación se explica el funcionamiento de la fase de aprendizaje: primeramente se copian las instancias del fichero de datos de entrenamiento en otro, añadiendo las cabeceras preparadas para ser leídas por Weka y se guarda con la extensión ARFF que es el formato soportado. De este modo se obtiene el fichero "ejemplo.arff" con las mismas instancias que el fichero original, ver *Figura 15*.

```
@relation ejemplo.arff

@attribute e0 numeric
@attribute e1 numeric
@attribute e2 numeric
@attribute e3 numeric
@attribute e4 numeric
@attribute e5 numeric
@attribute e6 numeric
@attribute e7 numeric
@attribute e8 numeric
@attribute modoMario {0, 1, 2}
@attribute a0 {0, 1}
@attribute a1 {0, 1}
@attribute a2 {0, 1}
@attribute a3 {0, 1}
@attribute a4 {0, 1}
@attribute a5 {0, 1}
@attribute monedasPantalla numeric
@attribute monedasRecogidas numeric
@attribute foso {0, 1}
@attribute obstaculo {0, 1}
@attribute tiempoRestante numeric

@data

0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 199
0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 198
0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 198
0, 0, 0, 0, 0, 0, 25, 0, 0, 2, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 198
0, 0, 0, 0, 0, 0, 25, 0, 0, 2, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 198
…
```

Figura 15. Creación del fichero de datos para Weka: ejemplo.ARFF.

Seguidamente se ejecuta el proceso de clustering utilizando la técnica k-medias, concretamente el algoritmo SimpleKMeans de la herramienta Weka. Se realiza el aprendizaje y se guarda el modelo del clúster (fichero *.model) para su posterior uso en la ejecución del agente automático.

En este proceso se crean tantos ficheros como clústeres se hayan definido. En ellos se reparten las instancias dependiendo de la cercanía de la instancia a su centroide, asignándose cada instancia del fichero de entrenamiento a cada clúster y guardándose en el fichero correspondiente. Los centroides son los representantes de cada clúster, que se almacenan en el modelo y servirán para realizar consultas posteriores para conocer a qué clúster pertenece una instancia.

Una vez completado el aprendizaje, se hace una copia de los ficheros guardando únicamente la información de los estados. Se explica su motivo en la fase *ejecutar el agente automático*.

En resumen, las salidas en este proceso de aprendizaje son las siguientes:

- fichero de datos, en formato .ARFF,
- fichero con el modelo del clúster, en formato .model,
- ficheros de clústeres, dependiendo del número de clúster que el usuario ha elegido, con toda la información de la instancia,
- copia de los anteriores ficheros de clústeres, pero únicamente con información del estado de las instancias.

Siguiendo con el ejemplo de la ejecución, en el cual se especificó un número de clúster igual a 3, tras finalizar el aprendizaje los ficheros generados (salidas) son:

- “ejemplo.arff”, fichero de datos con cabeceras para Weka,
- “ejemplo.model”, fichero con la información del proceso de clustering,
- “ejemploC0.arff”, “ejemploC1.arff” y “ejemploC2.arff” con toda la información de las instancias,
- “ejemploC0EST.arff”, “ejemploC1EST.arff” y “ejemploC2EST.arff” únicamente con la información de los estados.

Se puede observar una comparativa de dos ficheros, con toda la información o únicamente con la información del estado en la *Tabla 3*.

“ejemploC1.arff”	“ejemploC1EST.arff”
@relation ejemplo.arff @attribute e0 numeric @attribute e1 numeric @attribute e2 numeric @attribute e3 numeric @attribute e4 numeric @attribute e5 numeric @attribute e6 numeric @attribute e7 numeric @attribute e8 numeric @attribute modoMario {0, 1, 2} @attribute a0 {0, 1} @attribute a1 {0, 1} @attribute a2 {0, 1} @attribute a3 {0, 1} @attribute a4 {0, 1} @attribute a5 {0, 1} @attribute monedasPantalla numeric @attribute monedasRecogidas numeric @attribute foso {0, 1} @attribute obstaculo {0, 1} @attribute tiempoRestante numeric @data 0,0,0,0,25,0,0,0,80,2,0,1,0,1,1,0,0,0,0,1,198 0,-24,-24,0,2,2,0,2,2,2,0,1,0,1,1,0,23,15,0,1,188 -24,0,-24,0,0,0,0,0,0,2,0,1,0,0,0,0,8,20,0,1,186 0,0,-62,0,0,0,80,0,0,1,0,1,0,1,1,0,2,26,0,1,185 …	@relation ejemplo.arff @attribute e0 numeric @attribute e1 numeric @attribute e2 numeric @attribute e3 numeric @attribute e4 numeric @attribute e5 numeric @attribute e6 numeric @attribute e7 numeric @attribute e8 numeric @data 0,0,0,0,25,0,0,0,80 0,-24,-24,0,2,2,0,2,2 -24,0,-24,0,0,0,0,0,0 0,0,-62,0,0,0,80,0,0 …

Tabla 3. Comparativa de ficheros de clúster creados por Clustering.java.

Una vez completado el aprendizaje se procede a ejecutar el agente automático, ver *Figura 16.* Se utilizarán los ficheros creados para elegir la acción a ejecutar automáticamente dependiendo del estado en que se encuentre el agente.

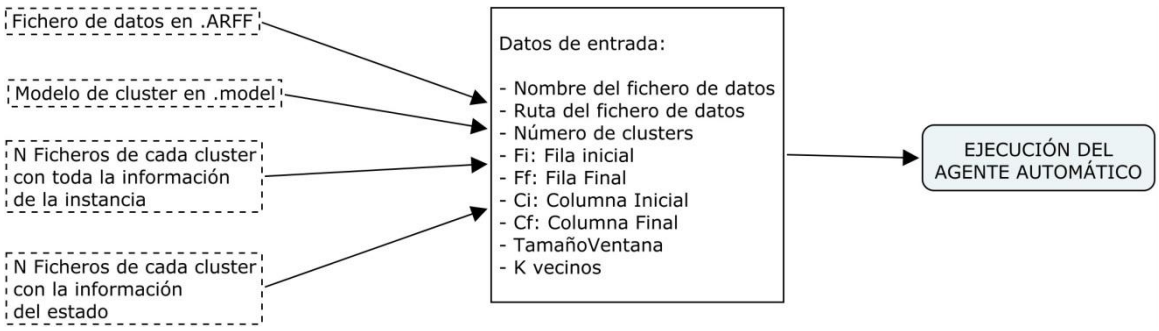

Figura 16. Ejecución del agente automático.

Primeramente se modifican los datos de entrada en el módulo “HAfinal.java”, ver *Figura 17.*

```
//Nombre Fichero
public static String datos = "ejemplo";
//Ruta del fichero
public static String ruta = "C:\\Desktop\\";
public static int numClusters = 3;
//Filas y Columnas (inicial y final) para la ventana de observación. Mario
siempre centrado en (9,9).
String ventana;
int Fi = 7;
int Ff = 9;
int Ci = 10;
int Cf = 12;
//Tamaño (M x N) de la ventana de Mario
public static int tamanoVentana = 9;
// K vecinos más cercanos
int K=2;
```

Figura 17. Código a modificar en HAfinal.java.

El usuario ha de definir en este módulo: la ruta y fichero de datos (en .ARFF) que se creó en la fase de aprendizaje, el número de clústeres que se usaron, la información del tamaño de la ventana y su tamaño (como un número entero). Solo falta definir el número K de vecinos más cercanos que se desean obtener del estado en el que se encuentre el agente automático para obtener así las acciones anteriormente ejecutadas en la fase de entrenamiento.

Para cambiar la configuración, ver *Figura 18,* se procede de la misma manera que en la fase de generación de datos de entrenamiento, seleccionando el proyecto *MarioAI, y* la clase principal *ch.idsia.scenarios.Main*

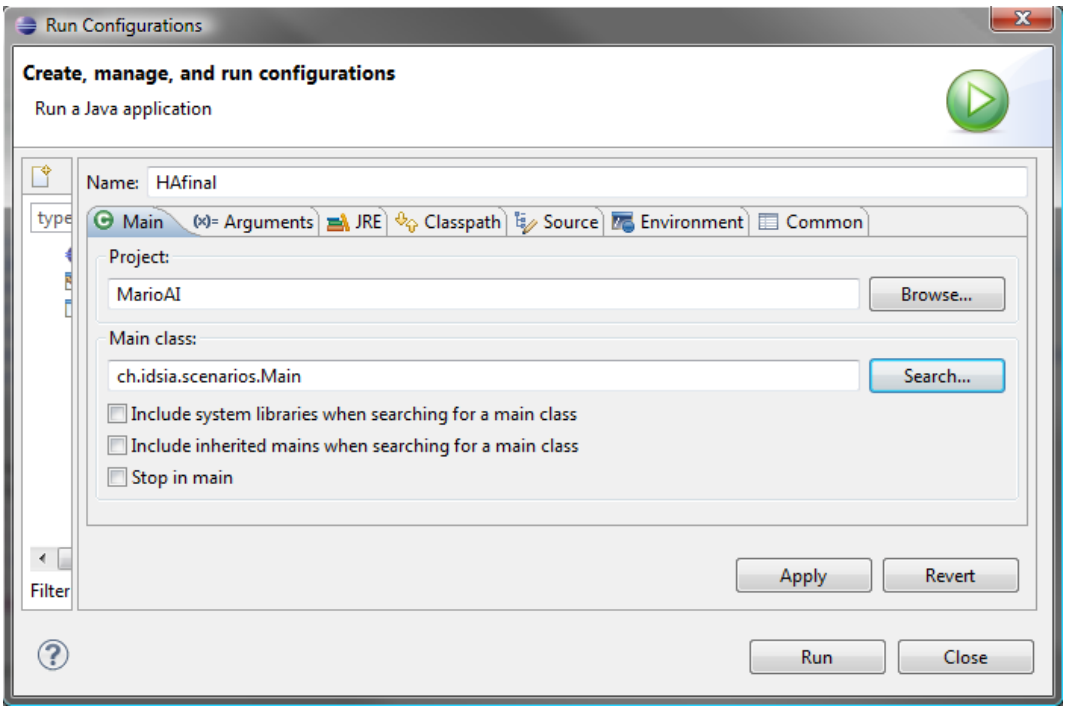

Figura 18. Cambio de configuraciones en Eclipse para HAfinal.java.

Igualmente es necesario cambiar el argumento para su ejecución, ver *Figura 19,* con *–ag.marioBros.agente.HAfinal.* En el capítulo 4, a la hora de realizar las pruebas, se añaden más *argumentos* para, por ejemplo, eliminar los enemigos de la partida.

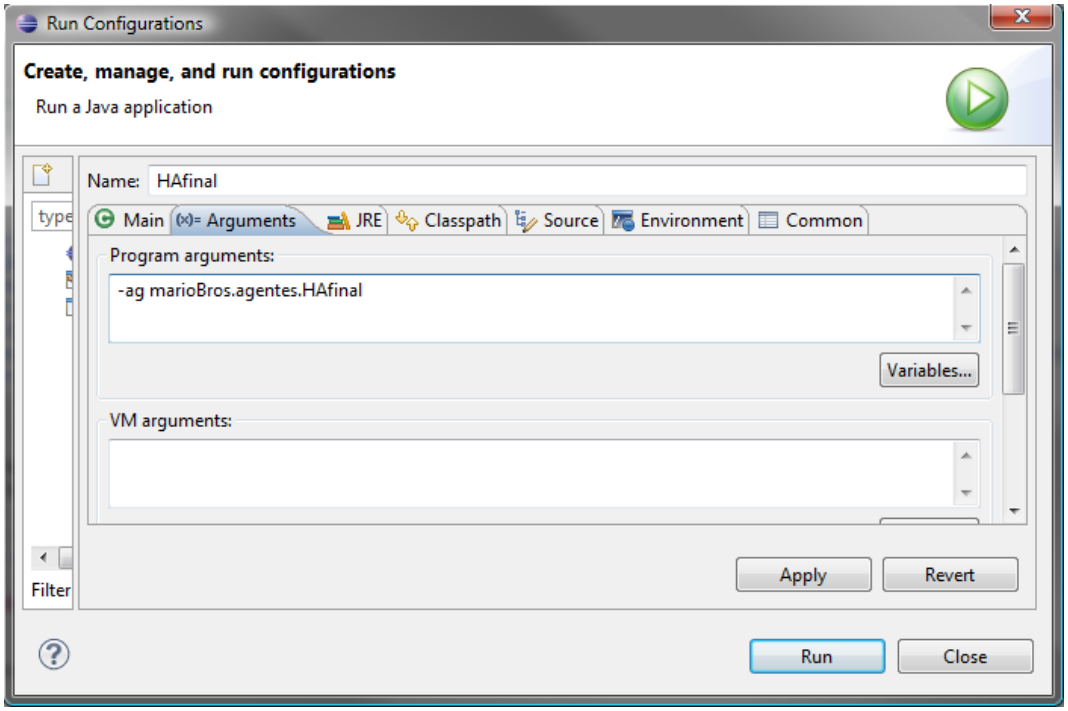

Figura 19. Cambio de argumentos en Eclipse para HAfinal.java.

Una vez todos los parámetros y variables se han definido, se procede a la ejecución del agente automático. Su funcionamiento es el siguiente:

- Primeramente se cargan en memoria todos los ficheros de clúster que se necesitan para realizar la búsqueda de información ya almacenada en la fase de generación de datos y en la fase de aprendizaje.

  Cada cierto tiempo, tic o instante (se trata del tiempo que tarda el programa MarioAI en ejecutar el código de la acción, que puede ser cambiado por el usuario en el módulo "HAfinal.java"), se consulta la acción que ha de ejecutar el agente automático en el estado actual.

  Aumentar este pequeño tiempo o tic es necesario ya que el agente necesita un margen temporal para realizar la acción. Se debe a que si se realiza cada tic una acción diferente esta no se verá ejecutada instantáneamente. Sucede de la misma manera si el usuario, mientras juega, pulsa las teclas Derecha+Saltar y las mantiene presionadas. Es necesario que las deje de pulsar para ejecutar una nueva acción, como podría ser Izquierda+Saltar.

- Para obtener la acción es necesario obtener el clúster al que pertenece el estado mediante el modelo que se genera a la hora de realizar clustering. Weka proporciona una función para tal fin.

- Una vez se tiene el número de clúster al que pertenece el estado actual del agente automático se necesita obtener el fichero de clúster que contiene únicamente la información del estado. De tal manera que en el ejemplo que se desarrolla en éste capítulo, al tener tres clústeres y tres ficheros generados con nombres "ejemploC0EST.arff", "ejemploC1EST.arff" y "ejemploC2EST.arff", si el resultado fuese que el estado pertenece al clúster 3, el fichero elegido será "ejemploC2EST.arff".

- Se realiza una búsqueda por estados en el fichero de clúster con solamente los estados, para que la búsqueda de los vecinos sea más acertada. La búsqueda de los mismos en el fichero de clúster que contiene toda la información de la instancia sería menos precisa.

  Se procede seguidamente a realizar una búsqueda de los K vecinos más cercanos, siendo K el número definido por el usuario. Para esta tarea se utiliza el algoritmo KNN de la herramienta Weka. A la hora de buscar los K vecinos únicamente por estado estos son almacenados en una lista. Para poder obtener su acción (o resto de información si fuera necesaria) iremos al mismo fichero de clúster, el cual contiene toda la información de las instancias (en el ejemplo, el fichero "ejemploC2.arff"). Se realizará una búsqueda por estados de los vecinos que se almacenan en él y se irán guardando las acciones que correspondan a éstos estados sin que las acciones se repitan. Se consigue de este modo una lista final de acciones que corresponden a los K vecinos.

- Por último, se pasan las acciones al agente para que ejecute la primera de ellas. Si con su ejecución el agente detecta, tras ciertos tics, que Mario sigue en la misma posición (se produce una situación de bloqueo del agente) se procede a ejecutar la siguiente acción de la lista, en caso de existir. Debido a que puedan existir situaciones de bloqueo de Mario, es posible que se recorra la lista de K acciones ejecutando cada vez una de ellas. Si no se consigue tras esto que Mario avance, entonces se realiza la acción por defecto "Derecha+Correr+Saltar" ya que el objetivo del juego es: llegar a la meta que siempre está a la derecha, existen obstáculos y se agota el tiempo.

Tras una ejecución completa del agente automático, el programa MarioAI nos dará los resultados de la partida en la consola de Eclipse. Siguiendo con el ejemplo, tras una ejecución del agente automático se obtiene el resultado de la *Figura 20.* Los parámetros de la salida en los que se centra el estudio de los experimentos son el tiempo transcurrido de la ejecución y el porcentaje superado del nivel, parámetros resaltados en dicha figura.

```
[MarioAI] ~ Evaluation Results for Task: BasicTask
       Evaluation lasted : 93825 ms
        Weighted Fitness : 5531
        Mario Status : Loss...
        Mario Mode : small
       Collisions with creatures : 3
       Passed (Cells, Phys) : 159 of 256, 2555 of 4096 (62% passed)
       Time Spent(marioseconds) : 33
       Time Left(marioseconds) : 167
       Coins Gained : 63 of 285 (22% collected)
       Hidden Blocks Found : 0 of 0 (0% found)
       Mushrooms Devoured : 0 of 1 found (0% collected)
       Flowers Devoured : 0 of 0 found (0% collected)
       kills Total : 12 of 40 found (30%)
       kills By Fire : 2
       kills By Shell : 0
       kills By Stomp : 10
       PunJ : 0
MEMO INFO: Reason of death: Collision with a creature [Goomba]

min = 2.0
max = 2.0
ave = 2.0
sd  = NaN
n   = 1
```

Figura 20. Salida de HAfinal.java.

3.2 Software necesario

Para la realización del proyecto se han utilizado los siguientes programas: Eclipse, JDK Java, Weka y MarioAI.

Para la programación en Java se ha utilizado el entorno de desarrollo integrado Eclipse. Se trata de un programa informático compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar aplicaciones. La definición que da el proyecto Eclipse acerca de su software es: "*una especie de herramienta universal, un IDE abierto y extensible para todo y nada en particular*".

También es necesario el paquete de software de desarrollo JDK de Java, que se compone de herramientas de desarrollo para la creación de programas en este lenguaje.

En lo referente al tratamiento de los datos, Weka es un software que contiene una colección de algoritmos de aprendizaje automático para tareas de minería de datos. Los algoritmos bien se pueden aplicar directamente a un conjunto de datos o realizar llamadas desde código Java como se ha realizado en este proyecto. Weka contiene herramientas para preprocesamiento de datos, clasificación, regresión, clustering, reglas de asociación y visualización.

La interfaz de programación de aplicaciones, API, que nos ofrece el proyecto de MarioAI y Weka, serán utilizados desde Eclipse.

3.3 Instalación

- Descargar e instalar el paquete "JAVA JDK 7", disponible en la página web de Oracle *[10]*. En éste proyecto se ha utilizado la versión "Java JDK 7u1 Windows i586".

- Proceder a la instalación del entorno de desarrollo. Se puede acceder directamente a la página web de Eclipse *[11]*. En este proyecto se utiliza la versión "Indigo SR1 win 32".

- Desde la página web del proyecto MarioAI, se obtiene el API de Mario *[12]*.

- El API de Weka *[13]*. Se ha utilizado internamente los algoritmos en el código.

Una vez se tienen todos los ficheros e instalados los programas, Java JDK y Eclipse, descomprimir el API de MarioAI y de Weka.

Con el programa Eclipse, abrir un proyecto nuevo importando toda la información de MarioAI.

El último paso es añadir las librerías de Weka para poder utilizarlas en Eclipse. Para ello basta con hacer clic derecho sobre el nuevo proyecto de Eclipse → Build Path → Configure Build Path → En el apartado Librerías hay que añadir el archivo .JAR de Weka.

Una vez hecho esto, todo está preparado para poder utilizar MarioAI y Weka en el IDE de Eclipse.

Capítulo 4

Diseño de experimentos y resultados

En el siguiente capítulo se pueden ver los experimentos que se han realizado con los diferentes valores utilizados y se proporcionan sus resultados con gráficas y tablas con una explicación sobre las mismas.

4.1 Introducción

Con motivo de los numerosos parámetros que pueden modificarse, este estudio se centra en los siguientes tres parámetros considerando el propósito de aprendizaje basado en instancias y el uso de técnicas de clustering:

- Tamaño del clúster. Se han tomado los siguientes valores: 2, 3, 4, 6, 8, 10 y 16.
- K (número de vecinos más cercanos). Varían entre 1, 2 y 3.
- Tamaño de la ventana. Se han tomado tres tamaños: 5x5, 2x7 y 3x3.

Debido a que el código de MarioAI está diseñado para una competición, éste limita el tiempo de consulta para la realización de la acción mediante la variable "COMPUTATION_TIME_BOUND", que estipula un determinado tiempo para realizarla, y que se encuentra en módulo *BasicTask.java* del API de MarioAI. Como consecuencia de las numerosas consultas y búsquedas que se realizan en este proyecto, es necesario aumentar ese valor (por defecto a 42, se ha aumentado a 1000) para poder realizar las pruebas y que en la salida por consola del resumen de la ejecución del agente se arrojen los resultados completos y no un fallo en el programa debido a esta causa.

El juego MarioAI permite la configuración de ciertos parámetros, como dificultad, longitud del nivel, etc. A continuación, se exponen las opciones del nivel que pueden ser utilizadas como argumentos:

- -vis off: Visualización del juego. Permite seguir visualmente la ejecución del agente automático. Los posibles valores son "on" para activarla y "off" para desactivarla.

- -ld 0: Dificultad del nivel. Entre 0 y 12. También se puede cambiar en el módulo "Main.java" como se explica en la *Figura 8*.

- -ls 0: Semilla del nivel. Igualmente posible cambiar en "Main.java", ver *Figura 8*.

- -lt 0: Tipo de Nivel. Entre 0 y 3, siendo por orden: superficie, subterráneo, castillo y aleatorio.

- -ll 300: Longitud del nivel.

- -le off: Los enemigos no aparecen en la partida. Se utiliza este parámetro en los ejemplos para eliminar los enemigos y que Mario recorra el nivel libremente.

Los anteriores parámetros pueden añadirse como argumentos al igual que en la *Figura 19*. Los usados en las pruebas son *–vis off* para que la ejecución sea más rápida al no mostrar cómo se supera la partida gráficamente y *-le on/off* para eliminar los enemigos o no del juego.

En siguientes apartados se explican las diferentes pruebas, mostrando el porcentaje de nivel superado junto con el tiempo de ejecución del agente para las configuraciones que se han elegido. Los resultados se muestran como una media del número de partidas jugadas por el agente.

Los ficheros de datos de entrenamiento se generan jugando 20 partidas por cada prueba, con la misma configuración de ventana, nivel, dificultad y semilla, con los que luego se ejecuta el agente automático con diferentes números de clúster y posteriormente con diferentes números de vecinos (K). Como resumen se tiene el siguiente esquema de la *Figura 21*.

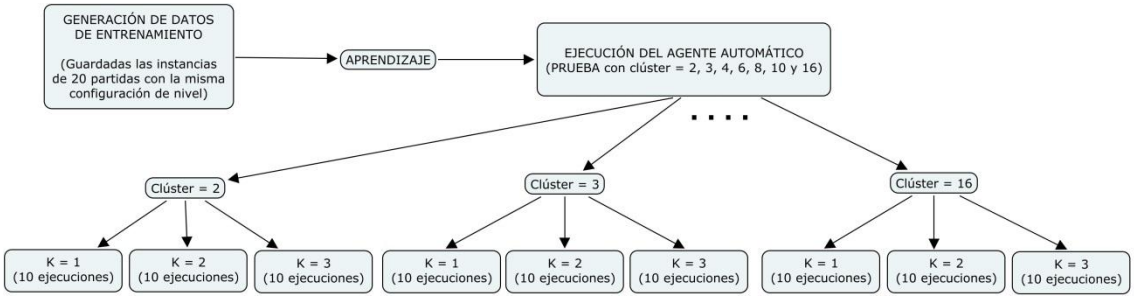

Figura 21. Resumen de una Prueba.

Primero se generan los datos de entrenamiento jugando 20 veces al mismo nivel, se realiza el aprendizaje y se crean los ficheros de clúster que se utilizan en cada una de las subpruebas (para diferentes números de clúster). Por cada clúster se realizan tres pruebas, con K (número de vecinos que se desean obtener) = 1, 2 y 3, y en cada una de ellas se ejecuta el agente 10 veces. Esto da lugar a 30 ejecuciones por cada clúster y en total se han elegido 7 diferentes números de clúster, con clúster = 2, 3, 4, 6, 8, 10 y 16. Por tanto se tiene un total de 210 ejecuciones. Al recrear la misma situación eliminando los enemigos del juego, en cada prueba se han realizado un total de 420 ejecuciones del agente automático para obtener los resultados que aparecen en los siguientes apartados.

4.2 Prueba 1: Ventana 5x5

A modo introductorio se explican todos los valores que detallan una prueba. En esta primera Prueba 1 se ha seleccionado una ventana de tamaño 5x5, cuyas filas y columnas tanto inicial como final son 7 y 11 respectivamente, lo que supone una ventana de valor 25 (= filas x columnas). Como se ha mencionado anteriormente, el número de partidas para la generación de datos de entrenamiento se extiende a 20. Se usan los valores por defecto para la configuración de nivel, semilla de nivel y dificultad. A la hora de aplicar SimpleKMeans de Weka es posible modificar la semilla y el número máximo de iteraciones del algoritmo. Debido a que pueden existir situaciones de bloqueo del agente automático, se irán realizando comparaciones de la posición de dicho agente en el eje x con posiciones anteriores en un intervalo de tiempo. Este intervalo es denominado Tic de bloqueo. Por último, Tic de acción se trata del intervalo de tiempo transcurrido hasta que una acción nueva es ejecutada, ya que el agente automático necesita un margen temporal para que la acción sea llevada a cabo. A continuación, ver *Tabla 4*, se detallan los valores de estas variables.

PRUEBA 1	
FILA INICIAL	7
FILA FINAL	11
COLUMNA INICIAL	7
COLUMNA FINAL	11
MATRIZ VENTANA	(5x5)
VENTANA	25
PARTIDAS ENTRENAMIENTO	20
NIVEL	0
SEMILLA DEL NIVEL	0
DIFICULTAD	0
SEMILLA SKMEANS	50
MAX.ITER.SKMEANS	100
TIC(BLOQUEO)	15
TIC(ACCION)	5

Tabla 4. Prueba 1: Configuración.

En esta prueba, con Mario centrado en la celda [9,9], se reúne información alrededor del personaje a cada dos celdas en cualquier dirección. Visualmente se ve representado en la *Tabla 5,* donde "M" es el personaje de Mario y se detallan las posiciones de las celdas.

7,7	*7,8*	*7,9*	*7,10*	*7,11*
8,7	*8,8*	*8,9*	*8,10*	*8,11*
9,7	*9,8*	**M**	*9,10*	*9,11*
10,7	*10,8*	*10,9*	*10,10*	*10,11*
11,7	*11,8*	*11,9*	*11,10*	*11,11*

Tabla 5. Prueba 1: Ventana 5x5.

Tras realizar la ejecución 10 veces por cada número de K y posteriormente por cada número de clúster, se presentan las medias de los resultados obtenidos con enemigos. En la *Tabla 6* se representa por cada clúster el nivel superado (%) y tiempo de ejecución. Los mismos datos se pueden ver detallados gráficamente en la *Figura 22* para el nivel superado y en la *Figura 23* para el tiempo de ejecución.

PRUEBA 1	K=1		K=2		K=3	
2 CLUSTERS	16%	523818.60	30%	70945.40	31%	808793.60
3 CLUSTERS	16%	192517.90	30%	46978.50	30%	104372.90
4 CLUSTERS	16%	198740.50	25%	44075.90	44%	90596.20
6 CLUSTERS	51%	30333.00	34%	41662.40	30%	62716.10
8 CLUSTERS	52%	20296.40	52%	78028.40	30%	43082.90
10 CLUSTERS	44%	18553.80	52%	50713.90	18.70%	32320.10
16 CLUSTERS	33%	10833.50	52%	37716.90	16%	303179.90
MEDIA TOTAL	**33%**	**142156.24**	**39%**	**52874.49**	**29%**	**206437.39**

Tabla 6. Prueba 1: Nivel superado y tiempo de ejecución con enemigos.

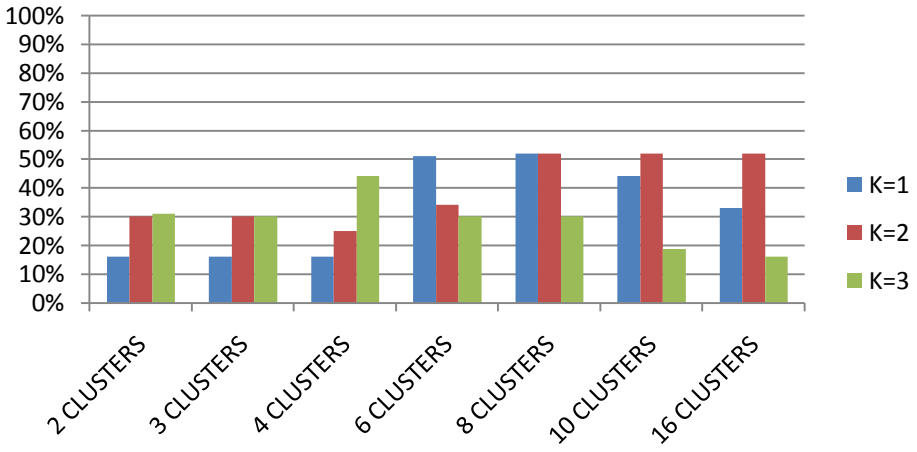

Figura 22. Prueba 1: Nivel superado con enemigos.

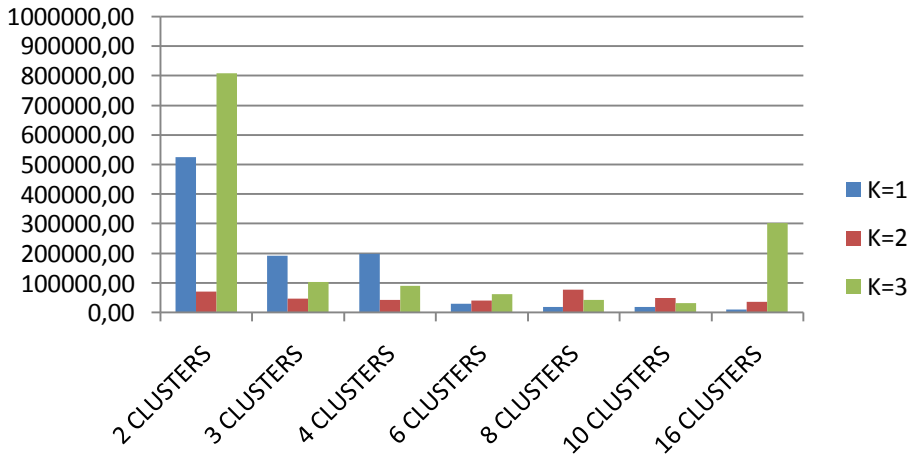

Figura 23. Prueba 1: Tiempo de ejecución con enemigos.

Como se puede observar en las anteriores figuras, en general, al aumentar el número de clúster se produce un ligero incremento del nivel superado y a la vez una disminución en el tiempo de ejecución. Los clústeres 6, 8 y 10 son los que menor tiempo de ejecución muestran e igualmente los que obtienen valores de porcentaje de nivel superado más altos y similares. Se puede observar que tanto en bajos como en muy altos números de clúster no se mejoran los valores anteriores. Si se enfoca el estudio en K número de vecinos, el valor K=2 es el que arroja mejores porcentajes en cuanto a valor medio total, es decir, es el que más distancia recorre en el nivel y además es el que se ejecuta en menor tiempo.

Como conclusión, se puede decir que el mejor valor de clúster es 8 y el mejor valor de K es 2 para la Prueba 1 en niveles con enemigos.

Ahora se realizan las mismas pruebas, con la misma configuración, pero eliminando los enemigos del juego. Se tienen así los resultados en la *Tabla 7,* y gráficamente en la *Figura 24* y *Figura 25* respectivamente.

PRUEBA 1	K=1		K=2		K=3	
2 CLUSTERS	100%	132955.00	100%	330489.40	100%	583128.30
3 CLUSTERS	100%	93141.30	100%	225288.60	100%	362056.80
4 CLUSTERS	100%	69423.50	100%	158628.20	100%	251606.10
6 CLUSTERS	100%	64891.6	100%	146262.40	100%	252343.50
8 CLUSTERS	100%	58571.80	100%	143975.80	100%	168785.80
10 CLUSTERS	100%	44361.70	100%	99038.00	100%	155205.80
16 CLUSTERS	100%	33674.70	100%	85944.90	100%	155205.80
MEDIA TOTAL	**100%**	**71002.80**	**100%**	**169946.76**	**100%**	**275476.01**

Tabla 7. Prueba 1: Nivel superado y tiempo de ejecución sin enemigos.

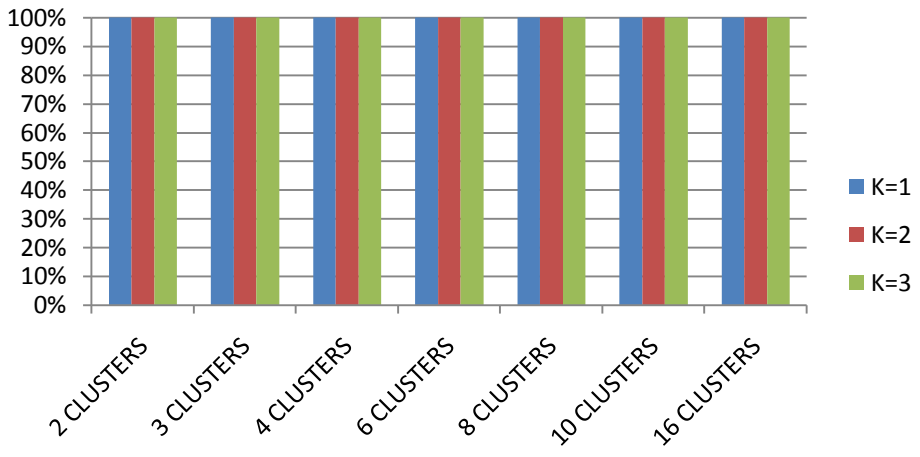

Figura 24. Prueba 1: Nivel superado sin enemigos.

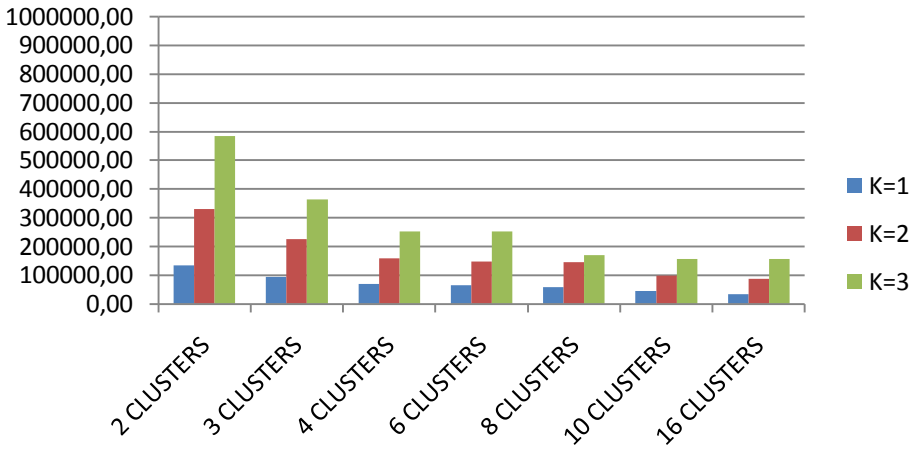

Figura 25. Prueba 1: Tiempo de ejecución sin enemigos.

En esta situación sin enemigos, se ha conseguido superar todas las partidas con éxito independientemente del número de clúster y K. En cuanto al tiempo de ejecución, a mayor número de clúster dicho tiempo disminuye. Las pruebas en las que ha tardado menos el agente automático en superar el nivel han sido las realizadas con K=1.

Como resumen a todo lo visto anteriormente, tanto con enemigos como sin ellos, se puede concluir que cuando no hay enemigos el agente automático ha conseguido llegar al final del nivel (nivel superado = 100%) y el tiempo transcurrido se ha reducido considerablemente en comparación a las pruebas realizadas cuando sí existen enemigos.

4.3 Prueba 2: Ventana 2x7

En la siguiente prueba se modifica el tamaño de la ventana para enfocar la información en las celdas anterior y posterior de Mario, como se observa en la *Tabla 8.* En este caso se tiene una visión horizontal del estado del agente automático, en vez de una visión más cuadrada como en la Prueba 1, para centrar el estado en los objetos y enemigos que hay a izquierda y derecha de Mario.

8,6	*8,7*	*8,8*	*8,9*	*8,10*	*8,11*	*8,12*
9,6	*9,7*	*9,8*	**M**	*9,10*	*9,11*	*9,12*

Tabla 8. Prueba 2: Ventana 2x7

La configuración para esta prueba es la misma que en la Prueba 1 (ver *más atrás*), con el único cambio de las variables que se refieren a la ventana: filas, columnas, matriz ventana y ventana. Se tienen por tanto los siguientes valores de la *Tabla 9:*

PRUEBA 2	
FILA INICIAL	8
FILA FINAL	9
COLUMNA INICIAL	6
COLUMNA FINAL	12
MATRIZ VENTANA	(2x7)
VENTANA	14
PARTIDAS ENTRENAMIENTO	20
NIVEL	0
SEMILLA DEL NIVEL	0
DIFICULTAD	0
SEMILLA SKMEANS	50
MAX.ITER.SKMEANS	100
TIC(BLOQUEO)	15
TIC(ACCION)	5

Tabla 9. Prueba 2: Configuración.

Se realiza el mismo estudio que anteriormente, con 10 ejecuciones por cada número de vecinos diferente y por cada número de clúster. Los resultados se pueden ver en la *Tabla 10,* junto con las respectivas *Figura 26* y *Figura 27* del nivel superado y tiempo de ejecución con enemigos.

PRUEBA 2	K=1		K=2		K=3	
2 CLUSTERS	17%	333028.10	33%	79151.50	33%	119898.80
3 CLUSTERS	17%	505578.80	33%	111961.10	44%	184383.60
4 CLUSTERS	17%	502490.30	33%	101933.40	44%	173662.80
6 CLUSTERS	17%	165066.80	30%	77854.50	44%	121104.00
8 CLUSTERS	17%	157550.00	30%	53299.20	44%	109338.80
10 CLUSTERS	17%	179392.30	33%	45479.70	44%	99253.50
16 CLUSTERS	17%	25764.70	33%	47618.00	44%	88033.20
MEDIA TOTAL	**17%**	**266981.57**	**32%**	**73899.63**	**42%**	**127953.53**

Tabla 10. Prueba 2: Nivel superado y tiempo de ejecución con enemigos.

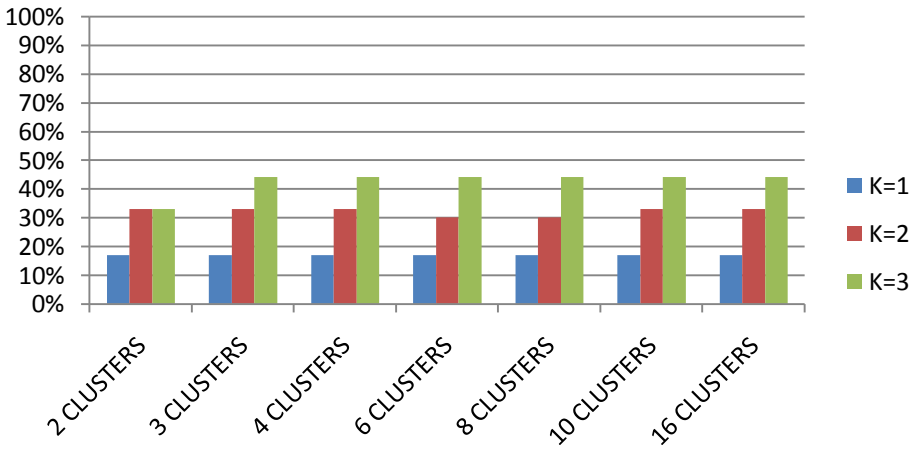

Figura 26. Prueba 2: Nivel superado con enemigos.

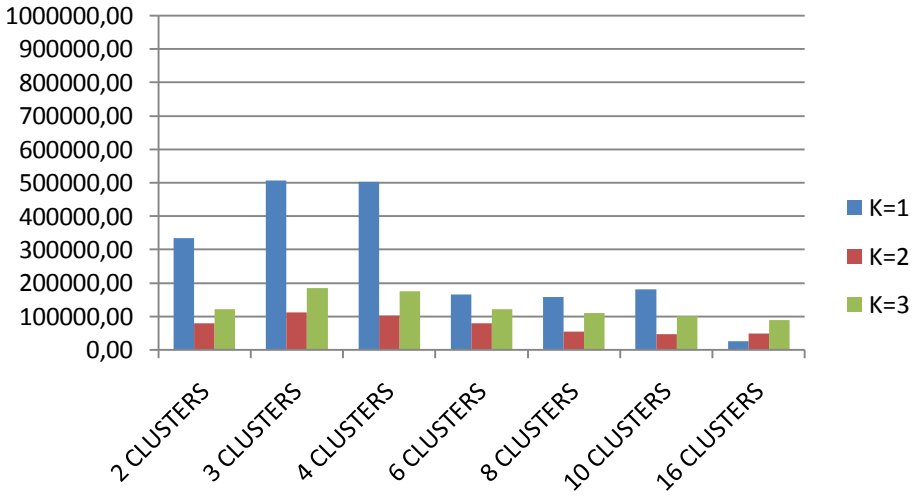

Figura 27. Prueba 2: Tiempo de ejecución con enemigos.

En relación al porcentaje de nivel superado, tras estos resultados se puede concluir que cuanto más crece el número de vecinos mayor es el porcentaje que se alcanza. La elección de un clúster u otro será indiferente ya que no hay apenas diferencias dentro de un mismo K. Comparado con los resultados en las mismas condiciones de la Prueba 1, las medias totales arrojan porcentajes más bajos excepto en K=3.

El tiempo de ejecución obtiene los mejores valores cuando K es 2, ya que es la menor en media. Al igual que en la Prueba 1 este valor de K es el que menos tiempo en ejecutarse tarda y aunque en esta Prueba 2 los mejores porcentajes de nivel superado sean para K=3, el doble de tiempo en éste no ha supuesto el doble del nivel superado. Observando la gráfica también se puede decir que a partir del clúster 6 se ha llegado a una normalidad en tiempos.

Igualmente en la *Tabla 11*, *Figura 28* y *Figura 29*, se observan los mismos resultados sin enemigos respectivamente.

PRUEBA 2	K=1		K=2		K=3	
2 CLUSTERS	17%	353418.30	42%	750263.60	60%	1192778.60
3 CLUSTERS	17%	370721.40	42%	686398.70	60%	1184457.90
4 CLUSTERS	17%	429032.30	42%	731095.70	60%	923993.10
6 CLUSTERS	17%	328419.60	42%	691935.00	100%	218790.10
8 CLUSTERS	17%	386100.00	100%	168709.60	100%	285175.20
10 CLUSTERS	17%	279093.90	42%	544731.20	42%	1043292.20
16 CLUSTERS	17%	27681.70	60%	572623.10	60%	807886.80
MEDIA TOTAL	**17%**	**310638.17**	**53%**	**592250.99**	**69%**	**808053.41**

Tabla 11. Prueba 2: Nivel superado y tiempo de ejecución sin enemigos.

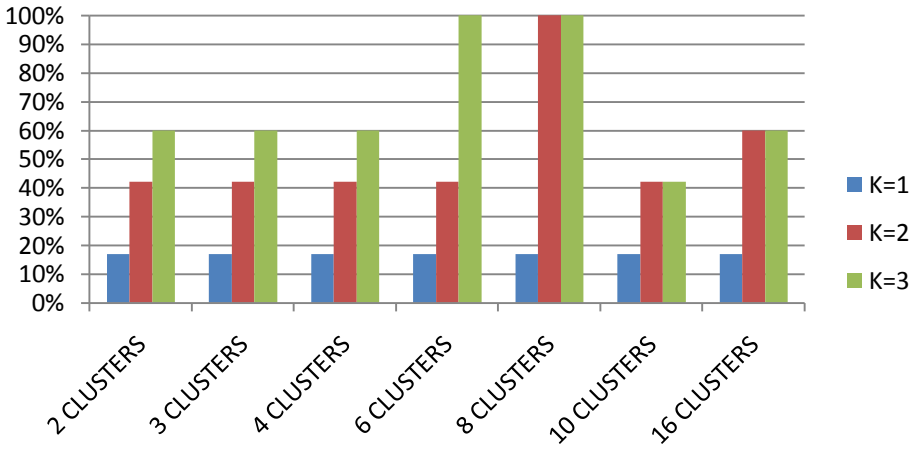

Figura 28. Prueba 2: Nivel superado sin enemigos.

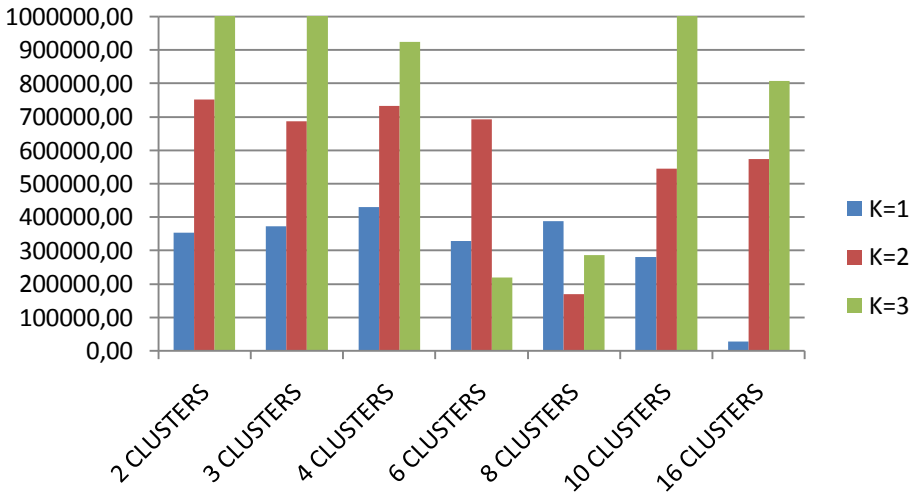

Figura 29. Prueba 2: Tiempo de ejecución sin enemigos.

En estos resultados sin enemigos se puede observar que K=2 y K=3 consumen mucho tiempo de ejecución en comparación con K=1. Los porcentajes de nivel superado no son del 100% como en la Prueba 1, pero se supera el 50% en dos de los tres vecinos. Cabe destacar que el clúster 8 ha dado el 100% y en el menor de los tiempos de ejecución para las Ks 2 y 3.

Como resumen, se alcanza aproximadamente el 50% de nivel superado tanto con enemigos como sin ellos (algo mejores en este caso), sin embargo el tiempo de ejecución es menor cuando no existen enemigos.

4.4 Prueba 3: Ventana 3x3

En la tercera prueba se ha optado por una ventana de tamaño 3x3, teniendo en cuenta lo siguiente que vería Mario si mirase al frente y arriba. Gráficamente puede verse en la *Tabla 12*. De este modo centramos la información del estado en la derecha del personaje y a una distancia de 3 casillas, que es donde realmente todo sucede ya que el juego se basa en desplazamiento lateral hacia esa dirección.

	7,10	*7,11*	*7,12*
	8,10	*8,11*	*8,12*
M	*9,10*	*9,11*	*9,12*

Tabla 12. Prueba 3: Ventana 3x3.

La configuración para esta prueba será igualmente la misma que en la Prueba 1, ver *más atrás*. Como se ha realizado en ambas pruebas anteriores, se han modificado las variables que se refieren a la ventana: filas, columnas, matriz ventana y ventana. Los valores de la configuración son los de la *Tabla 13:*

PRUEBA 3	
FILA INICIAL	7
FILA FINAL	9
COLUMNA INICIAL	10
COLUMNA FINAL	12
MATRIZ VENTANA	(3x3)
VENTANA	9
PARTIDAS ENTRENAMIENTO	20
NIVEL	0
SEMILLA DEL NIVEL	0
DIFICULTAD	0
SEMILLA SKMEANS	50
MAX.ITER.SKMEANS	100
TIC(BLOQUEO)	15
TIC(ACCION)	5

Tabla 13. Prueba 3: Configuración.

A continuación se muestran los resultados en la *Tabla 14,* y gráficamente en la *Figura 30* y la *Figura 31,* siendo los datos de nivel superado y tiempo de ejecución respectivamente, con enemigos.

PRUEBA 3	K=1		K=2		K=3	
2 CLUSTERS	44%	37324.60	22.8%	41168.30	53%	157819.60
3 CLUSTERS	44%	28573.70	22.8%	25743.90	62%	109873.80
4 CLUSTERS	47%	278172.00	44.8%	424148.50	34%	47682.40
6 CLUSTERS	44%	20818.00	22.8%	21749.50	34%	50181.20
8 CLUSTERS	34%	17163.70	44.8%	81142.00	34%	43454.80
10 CLUSTERS	34%	17564.60	44.8%	107293.60	34%	53491.00
16 CLUSTERS	58%	26465.90	45.7%	99911.70	34%	43370.80
MEDIA TOTAL	**44%**	**60868.93**	**36%**	**114451.07**	**41%**	**72267.66**

Tabla 14. Prueba 3: Nivel superado y tiempo de ejecución con enemigos.

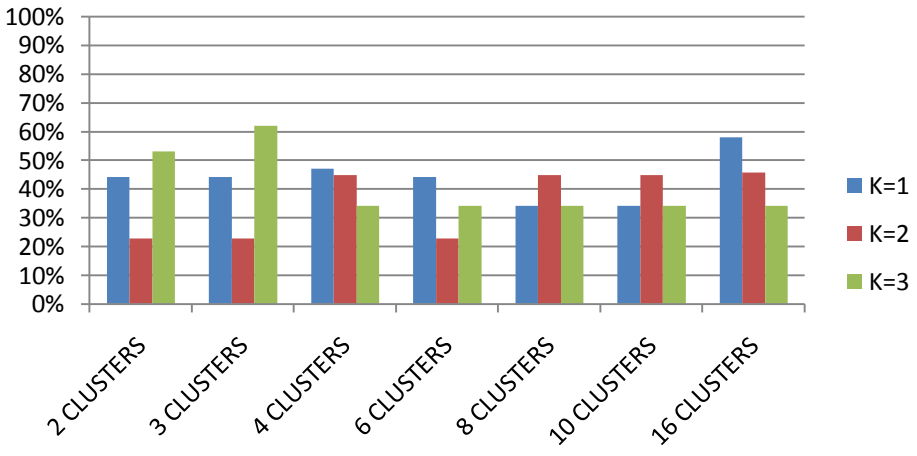

Figura 30. Prueba 3: Nivel superado con enemigos.

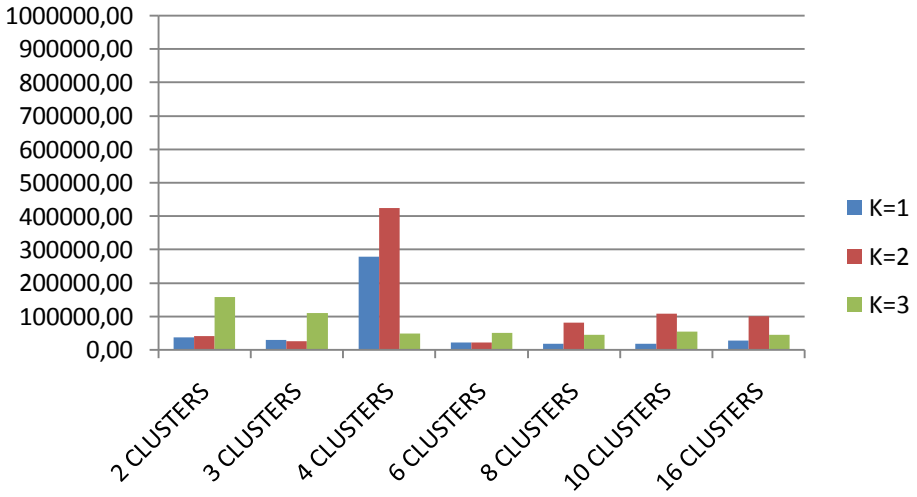

Figura 31. Prueba 3: Tiempo de ejecución con enemigos.

El porcentaje entre diferentes vecinos es bastante similar en media. Se puede observar que el nivel superado con enemigos en K=1 ha sido el más elevado de todas las pruebas realizadas. En lo referente al tiempo de ejecución, se han obtenido los valores más bajos en media de todas las pruebas posibles que se han realizado, además de arrojar los mejores porcentajes de nivel superado entre las tres pruebas realizadas con enemigos.

A continuación los resultados de la Prueba 3 para nivel superado y tiempo de ejecución sin enemigos. Ver *Tabla 15, Figura 32* y *Figura 33.*

PRUEBA 3	K=1		K=2		K=3	
2 CLUSTERS	51%	298241.90	41%	716220.50	17%	952429.50
3 CLUSTERS	58%	76686.30	41%	269482.50	17%	1154463.40
4 CLUSTERS	58%	58364.90	41%	84674.50	17%	1074132.80
6 CLUSTERS	58%	59564.90	41%	86877.30	17%	999598.80
8 CLUSTERS	58%	49865.90	41%	91214.30	100%	206948.00
10 CLUSTERS	58%	46241.20	41%	59408.00	100%	177605.70
16 CLUSTERS	17%	25012.50	58%	72433.80	100%	178104.80
MEDIA TOTAL	**51%**	**87711.09**	**43%**	**197187.27**	**53%**	**677611.86**

Tabla 15. Prueba 3: Nivel superado y tiempo de ejecución sin enemigos.

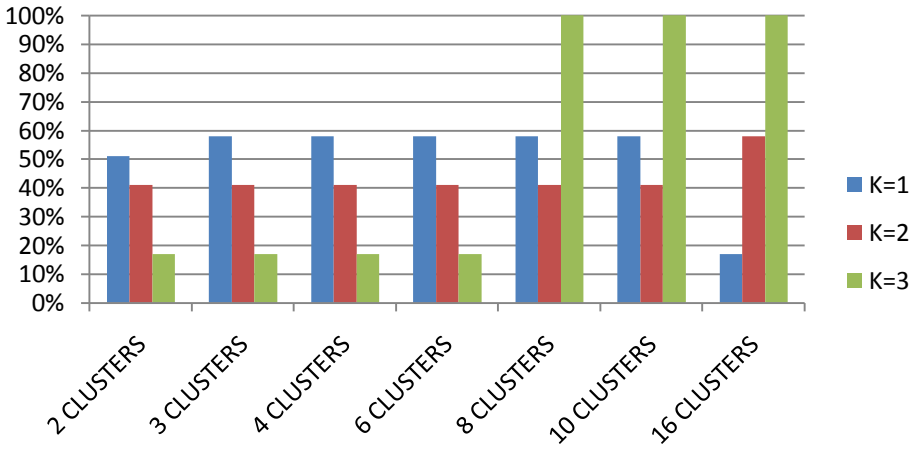

Figura 32. Prueba 3: Nivel superado sin enemigos.

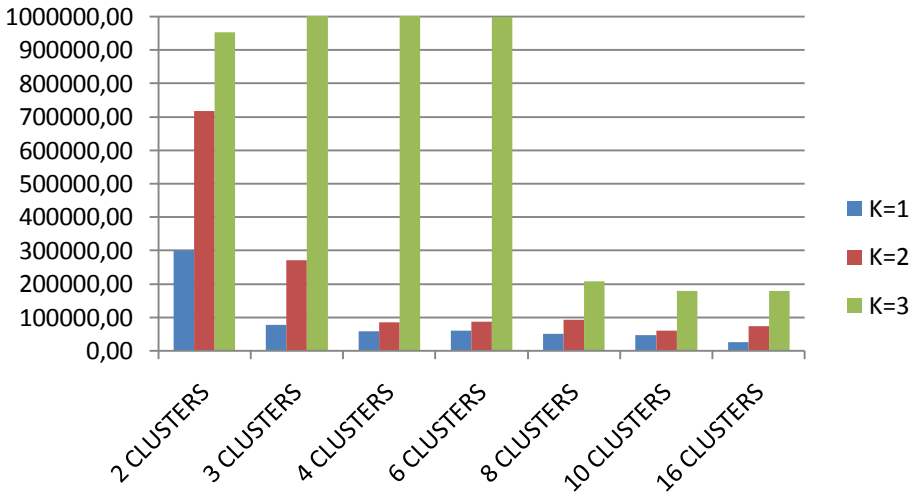

Figura 33. Prueba 3: Tiempo de ejecución sin enemigos.

Como análisis de la tercera prueba el porcentaje de niveles completados tanto con enemigos como sin ellos vuelve a estar cerca del 50%, siendo mucho menor el tiempo de ejecución cuando no hay enemigos. Destacar en esta situación que K=3 que obtiene el 100% cuando el número de clúster es mayor o igual a 8.

A continuación se detallan los resultados comparando todas las pruebas cuando hay enemigos o no, con los diferentes valores de los clústeres y los diferentes tipos de K (vecinos).

4.5 Resumen

Con los resultados de las pruebas se puede concluir que la mejor configuración ha sido la utilizada en la Prueba 1 (ventana de tamaño 5x5) ya que se consigue superar el 100% de la pantalla cuando no hay enemigos. En un escenario con enemigos, la mejor configuración ha resultado ser la Prueba 3 (ventana 3x3) ya que, como se ha mencionado *más atrás,* obtiene en media los porcentajes de nivel superado más altos de las tres pruebas además de tener los tiempos de ejecución más bajos.

A modo de resumen se muestran las siguientes tablas para las pruebas de nivel superado y tiempo de ejecución tanto con enemigos, ver *Tabla 16,* como sin enemigos, ver *Tabla 17*. Se tratan de las medias realizadas sobre los valores medios totales y agrupan los resultados de los vecinos más cercanos y de los clústeres en un único valor. Su propósito es conseguir una visión más resumida.

	NIVEL SUPERADO	TIEMPO DE EJECUCION
PRUEBA 1	33.46%	133822.70
PRUEBA 2	30.52%	156278.24
PRUEBA 3	39.93%	82529.22

Tabla 16. Resumen con enemigos.

	NIVEL SUPERADO	TIEMPO DE EJECUCION
PRUEBA 1	100.00%	172141.86
PRUEBA 2	46.24%	570314.19
PRUEBA 3	49.05%	320836.74

Tabla 17. Resumen sin enemigos.

Tras observar las dos tablas anteriores, se confirma lo anteriormente citado. Y es que la Prueba 3 con ventana de tamaño 3x3 y la Prueba 1 con ventana de tamaño 5x5 son los mejores porcentajes de nivel superado y tiempo de ejecución para niveles con enemigos y sin enemigo respectivamente.

También resulta interesante desglosar este resumen en cada tamaño de clúster, así se aprecia qué tipo de clúster ha sido el que mejores resultados ha proporcionado. El porcentaje de cada clúster se corresponde a la media de los mismos en los diferentes K. El clúster con mayor porcentaje ha sido el 8 en ambas tablas, con enemigos para la *Tabla 18,* y sin enemigos para la *Tabla 19.*

	PRUEBA 1	PRUEBA 2	PRUEBA 3	MEDIA
2 CLUSTERS	25.67%	27.67%	39.93%	31.09%
3 CLUSTERS	25.33%	31.33%	42.93%	33.20%
4 CLUSTERS	28.33%	31.33%	41.93%	33.87%
6 CLUSTERS	38.33%	30.33%	33.60%	34.09%
8 CLUSTERS	44.67%	30.33%	37.60%	37.53%
10 CLUSTERS	38.23%	31.33%	37.60%	35.72%
16 CLUSTERS	33.67%	31.33%	45.90%	36.97%

Tabla 18. Resumen por clúster con enemigos.

	PRUEBA 1	PRUEBA 2	PRUEBA3	MEDIA
2 CLUSTERS	100.00%	39.67%	36.33%	58.67%
3 CLUSTERS	100.00%	39.67%	38.67%	59.44%
4 CLUSTERS	100.00%	39.67%	38.67%	59.44%
6 CLUSTERS	100.00%	53.00%	38.67%	63.89%
8 CLUSTERS	100.00%	72.33%	66.33%	79.56%
10 CLUSTERS	100.00%	33.67%	66.33%	66.67%
16 CLUSTERS	100.00%	45.67%	58.33%	68.00%

Tabla 19. Resumen por clúster sin enemigos.

Por último, se comparan los porcentajes con diferentes K en los todos los clústeres. El porcentaje de cada K se corresponde a la media de la misma en los diferentes clústeres. K = 3 ha sido el que mayor porcentaje obtiene tanto con enemigos como sin ellos, ver *Tabla 20* y *Tabla 21* respectivamente.

	PRUEBA 1	PRUEBA 2	PRUEBA 3	MEDIA
K 1	32.57%	17.00%	44.00%	31.19%
K 2	39.00%	32.00%	36.00%	35.67%
K 3	29.00%	42.00%	41.00%	37.33%

Tabla 20. Resumen por K con enemigos.

	PRUEBA 1	PRUEBA 2	PRUEBA 3	MEDIA
K 1	100.00%	17.00%	51.00%	56.00%
K 2	100.00%	53.00%	43.00%	65.33%
K 3	100.00%	69.00%	53.00%	74.00%

Tabla 21. Resumen por K sin enemigos.

Como conclusión final a las pruebas, la mejor configuración media se obtiene con un tamaño de ventana de 3x3 para niveles con enemigos y de tamaño 5x5 para niveles sin enemigos, con 8 clústeres y con K = 3. Los resultados variarán entre unas pruebas y otras, pero generalmente se puede elegir esta configuración como la que mejores porcentajes y tiempo obtendrá en media.

Capítulo 5

Gestión del proyecto

Este capítulo trata sobre la planificación que se ha seguido para la elaboración del proyecto, así como un detalle del presupuesto definiendo los costes asociados al mismo.

5.1 Fases del desarrollo

- Motivación, definición y objetivos de los conceptos más importantes.
- Información sobre IA, AA y resto de algoritmos.
- Análisis del videojuego MarioAI y diseño.
- Aplicación de los conocimientos en la creación de un agente automático mediante técnicas de aprendizaje basado en instancias.

5.2 Medios empleados

Para la realización del proyecto se han utilizado los siguientes componentes:

- Ordenador: Intel Pentium Dual CPU 1.46GHz, 2GB RAM.
- Sistema operativo: Windows Vista.
- Java: Entorno de desarrollo (JDK).
- Eclipse: entorno de desarrollo integrado (IDE) de programación.
- Weka: programa para minería de datos. Se han utilizado sus algoritmos a través de Eclipse.
- Microsoft Excel: creación de tablas y gráficos.
- Microsoft Word: para la realización del presente documento.
- Microsoft Project: para la construcción del diagrama de Gantt.
- CmapTools: herramienta para la confección de los esquemas.

5.3 Planificación

Para documentar la planificación se utiliza la herramienta de software Microsoft Office Project a través de la cual se realiza el diagrama de Gantt, herramienta gráfica cuyo objetivo es mostrar el tiempo de dedicación previsto para diferentes tareas a lo largo de un tiempo total determinado. Para cada tarea, ver *Tabla 22.* Listado de tareas., se ha definido una duración que puede ser en días y horas, una fecha de comienzo, una fecha final y las tareas predecesoras, que indican qué tarea ha de finalizar antes de comenzar la tarea en cuestión.

Para la elaboración del diagrama se tiene en cuenta un horario de trabajo de lunes a viernes con una jornada laboral de tres horas, lo que supone un tiempo total de 437,25 días. Con una jornada laboral completa de 8 horas diarias, se estima una duración aproximada de 164 días. En la *Figura 34*se presenta el diagrama.

Id	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1	INICIO DEL PROYECTO	0 días	mar 29/11/11	mar 29/11/11	
2	**ESPECIFICACIÓN DE REQUISITOS**	**65,25 días**	**mar 29/11/11**	**mar 28/02/12**	
3	Inicio	0 días	mar 29/11/11	mar 29/11/11	1
4	Establecimiento objetivos generales	20 días	mar 29/11/11	lun 26/12/11	3
5	Análisis de herramientas	10 días	mar 27/12/11	lun 09/01/12	4
6	Requisitos de usuario y software	15 días	mar 10/01/12	lun 30/01/12	5
7	Estimación, viabilidad y planificación	20 días	mar 31/01/12	lun 27/02/12	6
8	Reunión	2 horas	mar 28/02/12	mar 28/02/12	7
9	Fin	0 días	mar 28/02/12	mar 28/02/12	8
10	**ANÁLISIS Y DISEÑO**	**65,5 días**	**mar 28/02/12**	**mar 29/05/12**	
11	Inicio	0 días	mar 28/02/12	mar 28/02/12	2
12	Análisis algoritmos	20 días	mar 28/02/12	mar 27/03/12	11
13	Estudio algoritmos y AA	16 días	mar 27/03/12	mié 18/04/12	12
14	Diseño del agente	25 días	mié 18/04/12	mié 23/05/12	13
15	Definición de pruebas	4 días	mié 23/05/12	mar 29/05/12	14
16	Reunión	4 horas	mar 29/05/12	mar 29/05/12	15
17	Fin	0 días	mar 29/05/12	mar 29/05/12	16
18	**IMPLEMENTACIÓN**	**251 días**	**mar 29/05/12**	**mié 15/05/13**	
19	Inicio	0 días	mar 29/05/12	mar 29/05/12	10
20	Fijación de objetivos	10 días	mar 29/05/12	mar 12/06/12	19
21	Revisión de diseño	5 días	mar 12/06/12	mar 19/06/12	20
22	Desarrollo	96,25 días	mar 19/06/12	mié 31/10/12	21
23	Evaluación	37 días	jue 01/11/12	vie 21/12/12	22
24	Reunión 1	2 horas	lun 24/12/12	lun 24/12/12	23
25	Implementación	50 días	lun 24/12/12	lun 04/03/13	24
26	Diseño de pruebas	15 días	lun 04/03/13	lun 25/03/13	25
27	Depuración	26 días	lun 25/03/13	mar 30/04/13	26
28	Reunión 2	2 horas	mar 30/04/13	mar 30/04/13	27
29	Pruebas finales	11 días	mar 30/04/13	mié 15/05/13	28
30	Reunión 3	2 horas	mié 15/05/13	mié 15/05/13	29
31	Fin	0 días	mié 15/05/13	mié 15/05/13	30
32	**MEMORIA**	**55,5 días**	**mié 15/05/13**	**jue 01/08/13**	
33	Inicio	0 días	mié 15/05/13	mié 15/05/13	18
34	Documentación	25 días	mié 15/05/13	mié 19/06/13	33
35	Informe final	20 días	mié 19/06/13	mié 17/07/13	34
36	Reunión 1	2 horas	mié 17/07/13	mié 17/07/13	35
37	Rediseño y mejoras	10 días	jue 18/07/13	mié 31/07/13	36
38	Reunión 2	2 horas	jue 01/08/13	jue 01/08/13	37
39	Fin	0 días	jue 01/08/13	jue 01/08/13	38
40	FIN DEL PROYECTO	0 días	jue 01/08/13	jue 01/08/13	39

Tabla 22. Listado de tareas.

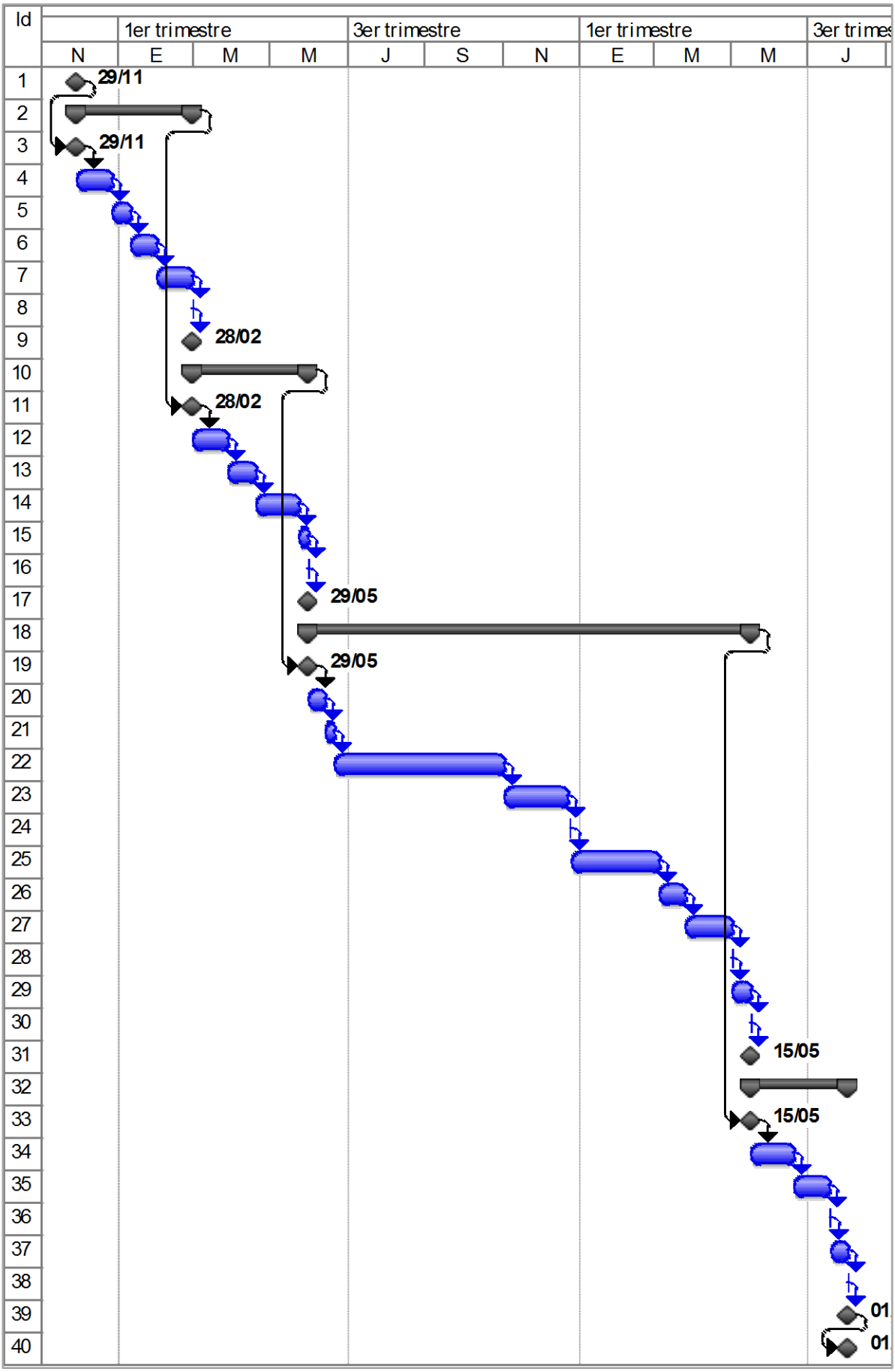

Figura 34. Diagrama de Gantt.

5.4 Presupuesto

Se detallan los diferentes costes divididos en costes de personal, costes de materiales y costes totales. Cada apartado se detalla con una tabla y una explicación de los mismos. Toda la información se realiza en base a una jornada de 8 horas y una duración del proyecto estimada en 164 días (5,4 meses), como se ha comentado en el apartado anterior.

Los recursos, ver *Tabla 23,* utilizados y su coste asociado son los siguientes:

Nombre	Categoría	Coste hombre mes (Euro)
RAFAEL LIAÑO RUIZ	Programador	1,500.00
FERNANDO FERNÁNDEZ REBOLLO	Jefe de proyecto 1	3,000.00
RAQUEL FUENTETAJA PIZÁN	Jefe de proyecto 2	3,000.00

Tabla 23. Tabla de recursos.

- Coste de personal: Las estimaciones de coste de los diferentes roles se indican en salarios brutos medios, es decir, con todos los impuestos incluidos. Ver *Tabla 24.*

Apellidos y nombre	Categoría	Dedicación	Coste hombre mes (Euro)	Coste (Euro)
LIAÑO RUIZ, RAFAEL	Programador	7	1,500.00	10,500.00
FERNANDEZ REBOLLO, FERNANDO	Jefe de Proyecto 1	2	3,000.00	6,000.00
FUENTETAJA PIZÁN, RAQUEL	Jefe de Proyecto 2	2	3,000.00	6,000.00
			Total	**22,500.00**

Tabla 24. Costes de personal.

- Coste de materiales: Para realizar el proyecto es necesario utilizar un ordenador. Esto implica disponer de las licencias necesarias para los diferentes programas y aplicaciones que se necesitan. Ver *Tabla 25.*

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable
Ordenador Portátil	500.00	100	5.4	60	45.00
Windows 8	119.99	100	5.4	60	10.80
Microsoft Office	539.00	100	5.4	60	48.51
Antivirus Panda	44.09	100	5.4	60	3.97
				Total	**108.28**

Tabla 25. Costes de materiales.

La fórmula de la amortización (coste imputable en *Tabla 25*) se calcula de la siguiente manera: $\left(\frac{A}{B}\right) x\ C\ x\ D$, dónde:

- A = nº de meses desde la fecha de facturación en que el equipo es utilizado.

- B = periodo de depreciación (60 meses).

- C = coste del equipo (sin IVA).

- D = porcentaje del uso que se dedica al proyecto (habitualmente 100%).

- Otros costes directos del proyecto o funcionamiento: En este apartado de gastos se incluyen todos los gastos no contemplados en los apartados anteriores. Ver *Tabla 26.*

Descripción	Costes imputable
Material de oficina	20.00
Electricidad	70.00
Internet	20.00
Consumibles	25.00
Transporte	60.00
Total	**195.00**

Tabla 26. Otros costes.

- Coste total: Resumen de todos los costes añadiendo los impuestos indirectos (20%). Ver *Tabla 27.*

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	22,500
Amortización	108
Costes de funcionamiento	195
Costes Indirectos (20%)	4,561
Total	**27,364**

Tabla 27. Coste total.

El presupuesto total de este proyecto asciende a la cantidad de **veintisiete mil trescientos sesenta y cuatro Euros.**

Capítulo 6

Conclusiones y trabajos futuros

En este capítulo se expone un breve resumen de la realización del proyecto con las conclusiones del mismo, valorando la consecución de los objetivos que se marcaron al principio, los problemas encontrados y una serie de trabajos futuros.

6.1 Conclusiones

El objetivo principal del proyecto es aplicar técnicas de aprendizaje basado en instancias para crear un agente automático, lo que ha llevado a estudiar y aprender diferentes apartados de la IA tales como algoritmos, técnicas y programas. Para cumplir el objetivo principal ha sido necesario:

- Estudiar el código proporcionado por la competición de MarioAI, para conocer internamente el código y su funcionamiento con los agentes.
- Diseñar la solución con técnicas de aprendizaje basado en instancias, para ello se ha estudiado teoría y ejemplos que no se han visto a lo largo de la carrera.
- Realizar experimentos para evaluar la aproximación y determinar que parámetros son los más adecuados, viendo de ésta manera como funciona mejor el agente.
- Documentar la aproximación con la presente memoria.

Siendo estos los objetivos que se propusieron al comienzo, se ha cumplido la realización de todos ellos. Se ha conseguido aplicar los conocimientos de IBL y clustering a una representación de un videojuego (en esta caso MarioAI), comprendiendo y aprendiendo su funcionamiento. El programa es capaz de almacenar las instancias en fichero y realizar consultas, aprendizaje y búsquedas sobre dichos datos organizados por clústeres. La integración de los algoritmos de Weka en el agente ha sido una tarea costosa ya que inicialmente se desconocía el funcionamiento del programa, pero tras un estudio previo y las diferentes explicaciones de los tutores ha resultado muy útil para aplicar su uso en el proyecto y conocer y aprender otras técnicas de agrupamiento y algoritmos.

El resultado final de las pruebas ha sido más satisfactorio cuando no se han utilizado enemigos. En la Prueba 1 se ha superado con éxito el nivel con todos los clústeres y números de vecinos más cercanos que se han ejecutado. Sin embargo, en situaciones con enemigos el agente automático no es capaz de llegar al final del nivel en ninguna ocasión. El máximo porcentaje logrado en este proyecto ha sido un 62%. Cuando se han utilizado niveles más complejos en los que existen fosos y enemigos, más bajo ha sido el porcentaje.

Como conclusión personal, lo más importante es que este proyecto ha servido para aprender nuevos conceptos, técnicas y algoritmos. En su realización se ha visto la utilidad de los sistemas de aprendizaje y se ha aumentado el interés en campos de la Inteligencia Artificial.

6.2 Limitaciones y trabajos futuros

Uno de los problemas ha sido la limitación de tiempo que tiene el código de la competición MarioAI para realizar la acción. En el proyecto se realizan numerosas comparaciones y búsquedas que hacen que se demore realizar la acción del agente automático. Para ello y como se comentó en el capítulo 4 *Diseño de experimentos y resultados*, hay que aumentar el valor de la variable que dicta ese tiempo.

La búsqueda del vecino más cercano podría hacerse más eficiente usando estructuras de datos inteligentes, como son los árboles kD (kD-trees en inglés). Al tratarse de una estructura en forma de árbol se puede acceder a datos cercanos más rápidamente. Otra posible solución interesante sería la optimización del código mediante el uso de tareas en paralelo para poder disminuir el tiempo de cálculo del agente automático.

En el proyecto se han creado diferentes variables que no se han usado, tales como la detección de un foso en el juego, de un obstáculo o el recuento de monedas. Se han mantenido para una posible ampliación, ya que al usar la técnica de IBL se han ignorado. El uso de estas variables podría ayudar al agente a no llegar a situaciones de bloqueo, evitar ser alcanzado por los enemigos y conseguir evitar los fosos. También puede ser útil generar unas instancias modelo que serían precargadas en el fichero antes de realizar la generación de datos. Esto podría solucionar algunos problemas con fosos y obstáculos ya que el agente tendría esa información almacenada. Otros posibles trabajos futuros serían el uso de otras técnicas de aprendizaje tales como A* o el aprendizaje por refuerzo.

Glosario

IA	*Inteligencia Artificial.*
AA	*Aprendizaje Automático.*
IBL	*Instance Based Learning.*
AI	*Artificial Intelligence.*
JDK	*Java Development Kit – Kit de Desarrollo Java.*
IDE	*Integrated Development Environment – Entorno de Desarrollo Integrado.*
MarioAI	*Mario Artificial Intelligence.*
CPU	*Central Processing Unit – Unidad central de procesamiento.*
GHz	*Gigahercio (10^9hercios).*
GB	*Gigabyte (10^9byte).*
Weka	*Waikato Environment for Knowlegde Analysis – Entorno para Análisis del conocimiento Universidad de Waikato.*
API	*Application Programming Interface.*
ARFF	*Atribute-Relation File Format.*
KNN	*K Nearest Neighbours – K vecinos más cercanos.*
NES	*Nintendo Entertainment System.*
IVA	*Impuesto sobre el Valor Añadido.*
JAR	*Java Archive – Archivo Java.*

Referencias

[1] Borrajo Millán, D., González Boticario, J., & Isasi Viñuela, P. *Aprendizaje Automático.* Sanz y Torres.

[2] Chomsky, N. (1998). *Nuestro conocimiento del lenguaje humano.* Chile: Ediciones Universidad de Concepción & Bravo y Allende Editores.

[3] Darwin, C. (1909). *El origen del hombre.* Valencia: Cuatro reales.

[4] Mira Mira, J., Esperanza Delgado, A., González Boticario, J., & Javier Díez, F. *Aspectos Básicos de la Inteligencia Artificial.* Sanz y Torres.

[5] Russell, S., & Norving, P. *Inteligencia Artificial: Un enfoque moderno.* Pearson, Prentice Hall.

[6] Pérez López, C., Santín González, D. (2007). *Minería de datos: técnicas y herramientas.* Thomson.

[7] http://es.wikipedia.org/wiki/Anexo:Videojuegos_de_Mario

[8] http://es.wikipedia.org/wiki/Anexo:Consolas_de_Nintendo#Lista_Cronol.C3.B3gica_de_las_Consolas_de_Nintendo

[9] http://www.marioai.org/

[10] www.oracle.com

[11] www.eclipse.org

[12] http://www.marioai.com/marioai-benchmark/download/MarioAI.zip

[13] http://prdownloads.sourceforge.net/weka/weka-3-7-9.zip

[14] https://sites.google.com/site/imarioproject/Home/zoom-levels-zlevels-

[15] http://clustering.jpmonge.com/

[16] Togelius, J., Karakovskiy, S., Koutník, J., Schmidhuber, J. *Super Mario Evolution.*

[17] http://aprendizajeuc3m.byethost12.com/aprendizaje-basado-en-instancias/

[18] http://julian.togelius.com/mariocompetition2009/index.php

[19] http://aigamedev.com/open/interview/mario-ai/

[20] http://www.doc.ic.ac.uk/~rb1006/projects:marioai

[21] http://weka.wikispaces.com/Use+WEKA+in+your+Java+code

[22] http://weka.wikispaces.com/Using+cluster+algorithms

[23] http://ccia.ei.uvigo.es/docencia/MRA/1011/practicas/api-weka/api-weka.html

[24] http://weka.wikispaces.com/Save+Instances+to+an+ARFF+File

[25] http://stackoverflow.com/questions/12118132/adding-a-new-instance-in-weka

[26] Rebollo, F. F., Millán, D. B. (27/02/2009). Aprendizaje basado en instancias. http://ocw.uc3m.es/ingenieria-informatica/aprendizaje-automatico/material-de-clase-1/aa-ocw-ibl.pdf

[27] http://www.ibm.com/developerworks/opensource/library/os-weka2/index.html?ca=drs-